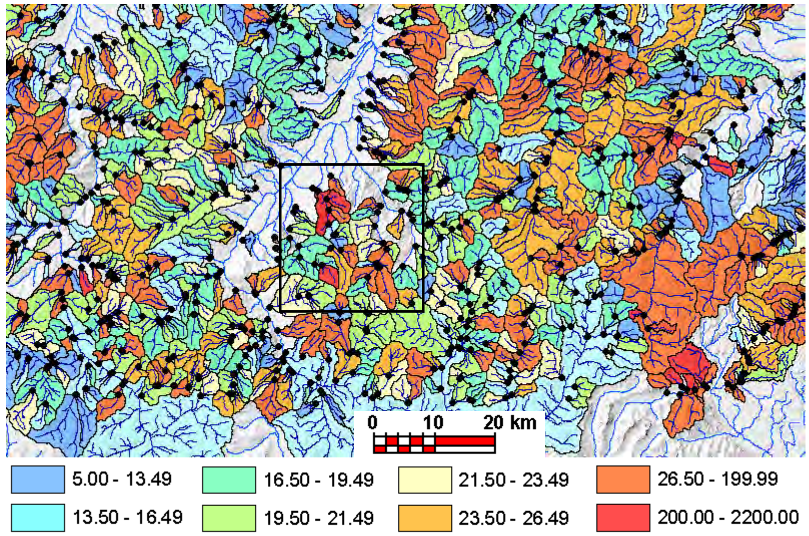
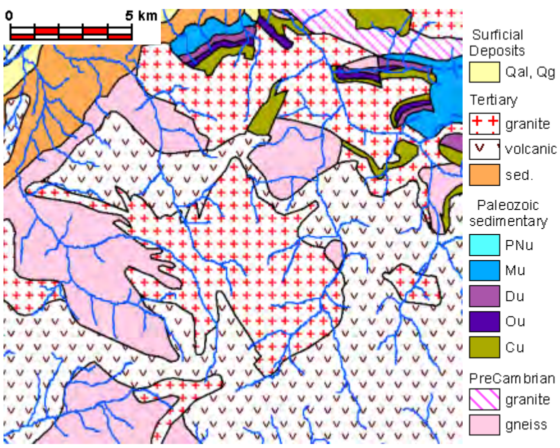
# Sample Script

# Catchment Analysis for Locating Ore Deposits

Regional surveys of stream sediment geochemistry are a major tool in exploration for mineral resources. Well-chosen sampling locations along major and tributary streams allow reconnaissance characterization of large areas and detection of anomalous elemental concentrations that could signal the presence of an ore body. The SampleCatchment script developed by MicroImages provides initial geospatial processing of geochemical datasets with hundreds or thousands of sample points and creates products that can be used as input for further geospatial and statistical analysis. The script uses a digital elevation model to delineate the upstream watershed catchment area for each sampling point and transfers the point attributes to the appropriate catchment polygons (see the color plate entitled *Sample Script: Mapping Catchment Areas for Sample Points*). Multiple samples within a single watershed generate subcatchments that are attributed to identify the number and identity of adjacent upstream and downstream catchments. The composition of each sample is influenced by all upstream subcatchments, so each catchment is also attributed with the total area of all contributing subcatchments.

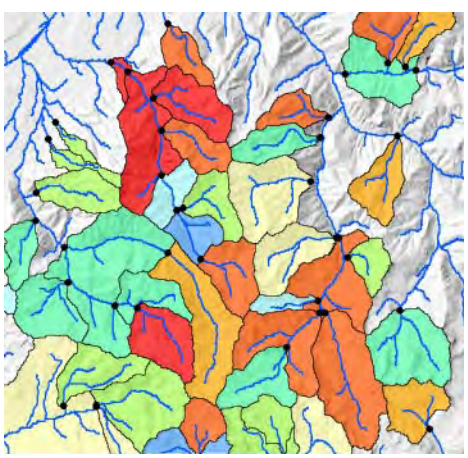



Catchments theme-mapped by copper concentration (ppm) in geochemical stream sediment samples taken at locations marked by black dots. Portion of a 26,000 square kilometer area with over 1200 sample points processed using the SampleCatchments script. Box outlines area shown in illustrations below.

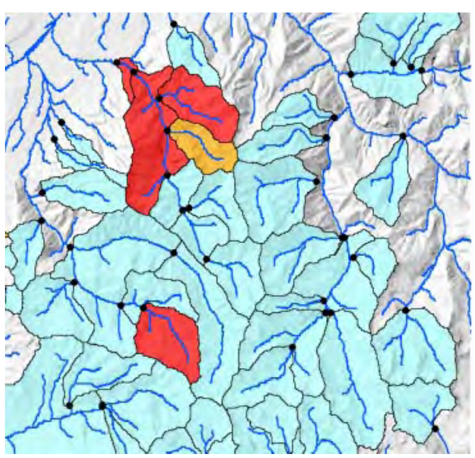



Geologic map of part of the test area.

The illustrations on this page provide a hypothetical example of how geospatial analysis of the sample catchments in TNTmips can be used to help solve one of the main interpretive challenges with geochemical data, the identification of anomalous concentrations. Background values of each element, such as copper, can fluctuate from catchment to catchment depending on the relative proportions of different rock types exposed in the contributing area, even in the absence of ore bodies. In order to predict background values of copper, the Polygon Properties process was used to overlay the catchments with a geologic map and determine for each subcatchment the areal extent and percentage of different rock units. A geospatial script (GeolUnitArea, excerpted on the opposite side of this page) was then used to identify for each catchment all of its contributing catchments and to sum the rock unit areas over those catchments. The resulting table of copper concentrations and unit percentages was used in a statistics program to perform a multilinear regression to compute a predicted copper concentration and residual for each catchment. After reimport to TNTmips, the residuals were then weighted by total contributing catchment area using a computed field and theme-mapped to reveal two catchments with anomalously high copper concentrations within the test area.

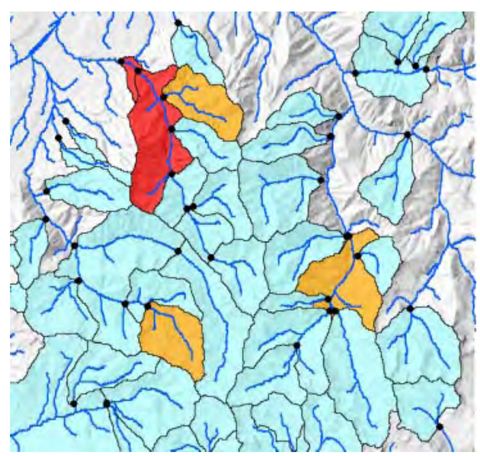


**Catchments Theme-Mapped by Measured Copper Concentration**
(legend same as upper right map)



**Residuals (measured minus predicted value) of $Log_{10}$ Copper Concentration (ppm)**

0.00 - 0.19 | 0.20 - 0.89 | 0.90 - 1.20



**Residuals of $Log_{10}$ Copper Concentration Weighted by Catchment Area**

0.00 - 1.99 | 2.00 - 19.99 | 20.00 - 61.84

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 GeolUnitArea.sml

Define recursive function to get upstream sample IDs from given sample

```
proc getUpstreamIDs ( string sampleID$ )
   {
   local numeric j;
   local string fieldbase$ = "UpSample";

   idList.AddToEnd(sampleID$);
```

add current sample ID to global ID List

get record number in PointToPoint table for current sample ID

```
   local numeric recNum =
         TableKeyFieldLookup(ptTable, "REC_NO", sampleID$);
```

get number of upstream samples immediately adjacent to the current basin

```
   local numeric numAdjUp =
         TableReadFieldNum(ptTable, "NumUpSamples", recNum);

   if ( numAdjUp > 0 )
      {
```

if there are adjacent upstream polygons, get list of sample IDs for adjacent upstream samples from UpSample[num] fields in PolyToPoly table

```
      local class STRINGLIST tempList;
      for j = 1 to numAdjUp
         {
```

create string for field name holding the appropriate UpSample basin ID and add to local stringlist

```
         local string field$ = fieldbase$ + NumToStr(j);
         tempList.AddToEnd( TableReadFieldStr(ptTable, field$, recNum) );
         }
```

loop through local list of UpSample fields to get their adjacent upstream basin IDs and call procedure to check for basins further upstream

```
      for j = 1 to numAdjUp
         {
         getUpstreamIDs( tempList[j-1] );
         }
      }
   }
```

loop through records in UnitArea table and sum area of each geologic unit in contributing area, compute percentage for each unit, and write values to the table

```
for i = 1 to unitAreaTbl.NumRecords
   {
   local numeric j, k, m, n;
   local string sampID$ = BasinVectOut.poly.UnitArea[@i].REC_NO$;

   local numeric recNum;
   local numeric sumBasinArea;
```

record number in PolyToPoly table

running sum of polygon areas for basin

```
   local numeric cuArea, duArea, muArea;
   local numeric cuPct, duPct, muPct;
```

numeric variables for summed area and percent of each rock unit in map area

array to hold element numbers of polygons PolyToPoly record is attached to

```
   local array numeric polyList[1];
   local numeric numAttachedPolys;
```

number of polygons attached

```
   local array numeric pctRecordList[1];
   local numeric numPctRecords;
```

array to hold record numbers of PERCENTAGE table records attached to each polygon

```
   getUpstreamIDs( sampID$ );
```

call recursive function to get list of all upstream sample/basin IDs

loop through list of contributing basins to sum geologic unit areas

```
   for j = 1 to idList.GetNumItems()
      {
```

get record number for basin's record in the PolyToPoly table

```
      recNum = TableKeyFieldLookup(basinInfoTbl, "REC_NO", idList[j-1]);
```

get list of polygon elements attached to that record

```
      numAttachedPolys =
      TableGetRecordElementList(basinInfoTbl, recNum, polyList);

   for k = 1 to numAttachedPolys
      {
```

loop through polygons

```
      sumBasinArea += BasinVectOut.poly[ polyList[k] ].POLYSTATS.Area;
```

get list of record numbers of PERCENTAGE table records attached to polygon

```
      numPctRecords =
            TableReadAttachment(percentTbl, polyList[k], pctRecordList);

      for m = 1 to numPctRecords
         {
         n = pctRecordList[m];
```

loop through attached records to compute sums

```
         if ( BasinVectOut.poly.PERCENTAGE[@n].FORMATION$ == "Cu")
            then cuArea += BasinVectOut.poly.PERCENTAGE[@n].Area;
         else
         if ( BasinVectOut.poly.PERCENTAGE[@n].FORMATION$ == "Du")
            then duArea += BasinVectOut.poly.PERCENTAGE[@n].Area;
         else
         if ( BasinVectOut.poly.PERCENTAGE[@n].FORMATION$ == "Mu")
            then muArea += BasinVectOut.poly.PERCENTAGE[@n].Area;

         [repeat for all rock units]
         }
      }
   }

cuPct = 100 * cuArea / sumBasinArea;
duPct = 100 * duArea / sumBasinArea;
muPct = 100 * muArea / sumBasinArea;
[repeat for all rock units]
```

use summed basin area and summed unit area to calculate percentage

sum unit percentages for each basin as check

```
totUnitPct = round( (cuPct + duPct + muPct + ouPct + qalPct + tgPct +
         tsPct + tvPct + pCgrPct + pCgsPct ) );

sumBasinArea = sumBasinArea / 1000000;
cuArea = cuArea / 1000000;
duArea = duArea / 1000000;
muArea = muArea / 1000000;
[repeat for all rock units]
```

convert summed areas from square meters to square km

write computed results to appropriate fields in UnitArea table

```
TableWriteField(unitAreaTbl, i, "CheckArea", sumBasinArea);

TableWriteField(unitAreaTbl, i, "Cu_Area", cuArea);
TableWriteField(unitAreaTbl, i, "Cu_Pct", cuPct);

TableWriteField(unitAreaTbl, i, "Du_Area", duArea);
TableWriteField(unitAreaTbl, i, "Du_Pct", duPct);

TableWriteField(unitAreaTbl, i, "Mu_Area", muArea);
TableWriteField(unitAreaTbl, i, "Mu_Pct", muPct);

[repeat for all rock units]

TableWriteField(unitAreaTbl, i, "Total_Unit_Pct", totUnitPct);
```

clear all sum variables for next pass

```
sumBasinArea = 0;
cuArea = 0;      duArea = 0;      muArea = 0;       ouArea = 0;
qalArea = 0;     tgArea = 0;      tsArea = 0;       tvArea = 0;
pCgrArea = 0;    pCgsArea = 0;    totUnitPct = 0;

idList.Clear();
}
```

clear global stringlist of contributing basin IDs