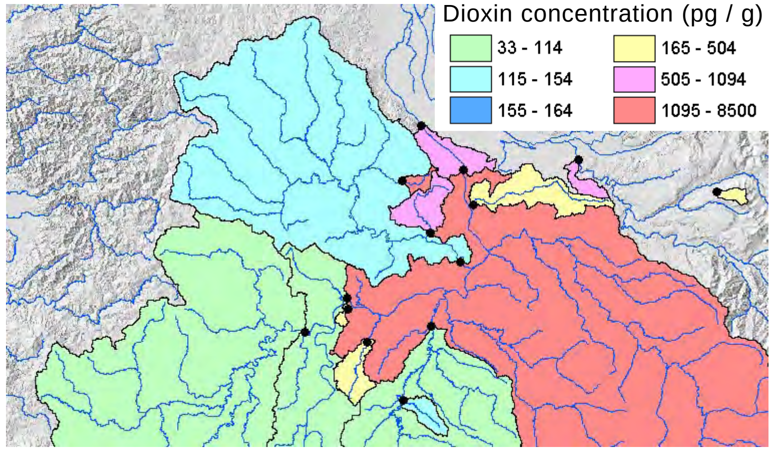
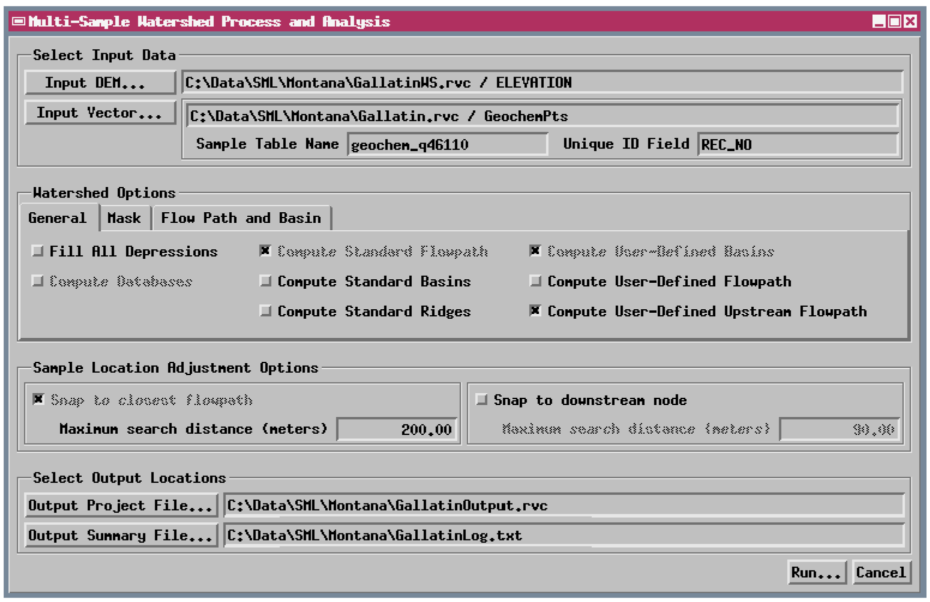
Sample Script

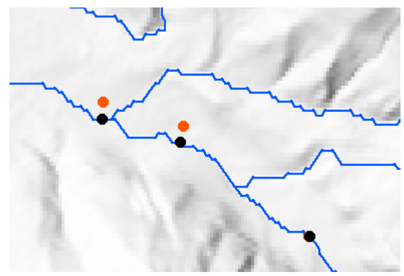
Mapping Catchment Areas for Sample Points

The SampleCatchments script developed by MicroImages demonstrates how the TNT geospatial scripting language (SML) enables you to automate complex processing of your geospatial data and provide custom analysis capabilities. This script uses functions and data structures associated with the TNTmips Watershed process to delineate the upstream watershed catchments for sample locations represented by many points in a vector object. The TNTmips Watershed process provides interactive manual placement of seed points to compute upstream catchment areas, but this is only practical for a few sample points. The SampleCatchments script automates this procedure for use with hundreds or thousands of sample points and also creates appropriate attributes for the derived catchments. The catchment mapped by the script for each sample point encompasses the area of the landscape that could have contributed to the composition of the water or stream sediment sample collected at the point's location.

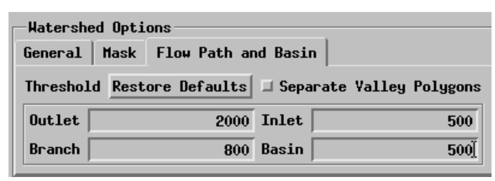


Stream sediment sampling locations (black dots) monitoring dioxin pollutant levels in the lower Willamette River basin (Oregon, USA). Upstream catchment polygons determined by the SampleCatchments script for these locations are theme-mapped by dioxin concentration (in picograms dioxin per gram of sample).





Input sample points (orange dots) may not lie exactly on the flowpaths calculated by the watershed function, so the script snaps each sample point to the closest vertex on the closest flowpath line (black dots) within the specified search distance before delineating the upstream catchments.



You can use either a raw DEM or a depressionless DEM previously created by the Watershed process. In the latter case, turn off the Fill All Depressions toggle button to skip this computationally-intensive step. Use the Flow Path and Basin panel to set the parameters controlling the density and complexity of the flowpaths created by the script. For the most efficient processing of many sample points, process the DEM through the standard Watershed process first and experiment there to determine the flowpath parameter set that creates the flowpath network with the minimum required density and complexity.

The vector objects and associated databases produced by this script could be used to help identify likely source areas associated with anomalous sample compositions in stream pollution studies and in mineral exploration surveys, among other applications.

Script inputs include an elevation raster object (DEM) and the vector object containing the sample points. These objects are selected and other script parameters are set using controls on a custom dialog window created and opened by the script.

The script, which is excerpted on the reverse side of this plate, produces a vector object containing the sample points after relocation to the nearest watershed flowpath and another containing the catchment area polygons, among others. Any database attributes attached to the sample points are automatically transferred to records attached to the relevant catchment polygons. The color plate entitled *Sample Script: Catchment Analysis for Locating Ore Deposits* provides additional examples of the use of this script.

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 SampleCatchments.sml

Use the new sample point locations to compute a new user defined watershed

```
string watershedparms$ = "";
if ( mainDLG.GetCtrlByID("id_FILL_ALL_DEP").GetValueNum()) {
    watershedparms$ += "FillAllDepressions,";
    }
if ( mainDLG.GetCtrlByID("id_COMP_USR_FLOWPATH").GetValueNum()) {
    watershedparms$ += "FlowPath,";
    }
watershedparms$ += "Basin";
WatershedComputeElements(watershed, x, y, numSamplePoints, watershedparms$);
```

Recreate the watershed elements using the new modified points

```
if ( mainDLG.GetCtrlByID("id_COMP_USR_FLOWPATH").GetValueNum()) {
    WatershedGetObject(watershed, "VectorUserFlowPath", ObjectFilename, ObjectName);
    ObjNumber = ObjectNumber(ObjectFilename, ObjectName, "VECTOR");
    CopyObject(ObjectFilename, ObjNumber, OutputProjectFile);
    }

WatershedGetObject(watershed, "VectorUserBasin", ObjectFilename, ObjectName);
ObjNumber = ObjectNumber(ObjectFilename, ObjectName, "VECTOR");
CopyObject(ObjectFilename, ObjNumber, OutputProjectFile);
```

Get a vector object handle for the computed basins from the watershed structure

```
OpenVector(UBasinVECT, OutputProjectFile, "USDBASIN");
local class GEOREF UBSNgeoref = GetLastUsedGeorefObject(UBasinVECT);
```

Open the "VectorUserBasin" and get it's georef

```
local numeric numBasinsComputed = 0;
local numeric numUserBasins = NumVectorPolys(UBasinVECT);
local class DATABASE UBasinDB = OpenVectorPolyDatabase(UBasinVECT);
```

Relate the user basins to the appropriate sample points that are within the basin

```
if (TableCopy(SampleDB, SampleTBL, UBasinDB) < 0) {
    message = "Error occured while trying to copy the sample table.\n" + "Exiting the script now... \n";
    print(message);
    fwritestring(summaryFile, message);

    CloseRaster(SampleDEM);      CloseVector(SampleVECT);
    CloseVector(WarpVECT);       CloseVector(ResultVECT);       CloseVector(UBasinVECT);
    Exit();
    }
```

Copy the sample database table from original vector

```
local class DBTABLEINFO UBasinSampleTBL = DatabaseGetTableInfo(UBasinDB, sampleTblName);
UBasinSampleTBL.OneRecordPerElement = 1;

local class DBTABLEINFO UBasinPolyToPolyTBL = TableCreate(UBasinDB, PolyToPolyTblName, PolyToPolyTblDesc);

TableAddFieldString(UBasinPolyToPolyTBL, sampleIDFldName, 12);
```

Sample ID related to the basin polygon

```
TableAddFieldInteger(UBasinPolyToPolyTBL, numPolysPerSampleFldName, 15);
```

Number of vector polygons related to that sample

```
TableAddFieldFloat(UBasinPolyToPolyTBL, areaSamplePolysFldName, 15, 4);
```

Total of all vector polygons related to that sample

```
for i = 1 to numUserBasins {
    records[1] = TableWriteRecord(UBasinPolyToPolyTBL, 0, "", 1, -1);
    TableWriteAttachment(UBasinPolyToPolyTBL, i, records, 1, "polygon");
    }
```

Create a record in the table for each basin

```
for i = 1 to numSamplePoints {
    Temp1ObjX = ResultVECT.Point[i].Internal.x;
    Temp1ObjY = ResultVECT.Point[i].Internal.y;
    ObjectToMap(ResultVECT, Temp1ObjX, Temp1ObjY, RSLTgeoref, Temp1MapX, Temp1MapY);

    polyNum = FindClosestPoly(UBasinVECT, Temp1MapX, Temp1MapY, UBSNgeoref);

    if (polyNum > 0) {
        TableReadAttachment(ResultSampleTBL, i, records, "point");
        sampleNum$ = TableReadFieldStr(ResultSampleTBL, sampleIDFldName, records[1]);

        TableReadAttachment(UBasinPolyToPolyTBL, polyNum, records, "polygon");
        TableWriteField(UBasinPolyToPolyTBL, records[1], sampleIDFldName, sampleNum$);
        TableWriteField(UBasinPolyToPolyTBL, records[1], numPolysPerSampleFldName, 1);
        }
    }
```

In object coordinates

If a polygon was found, save the sample name to the polygon table