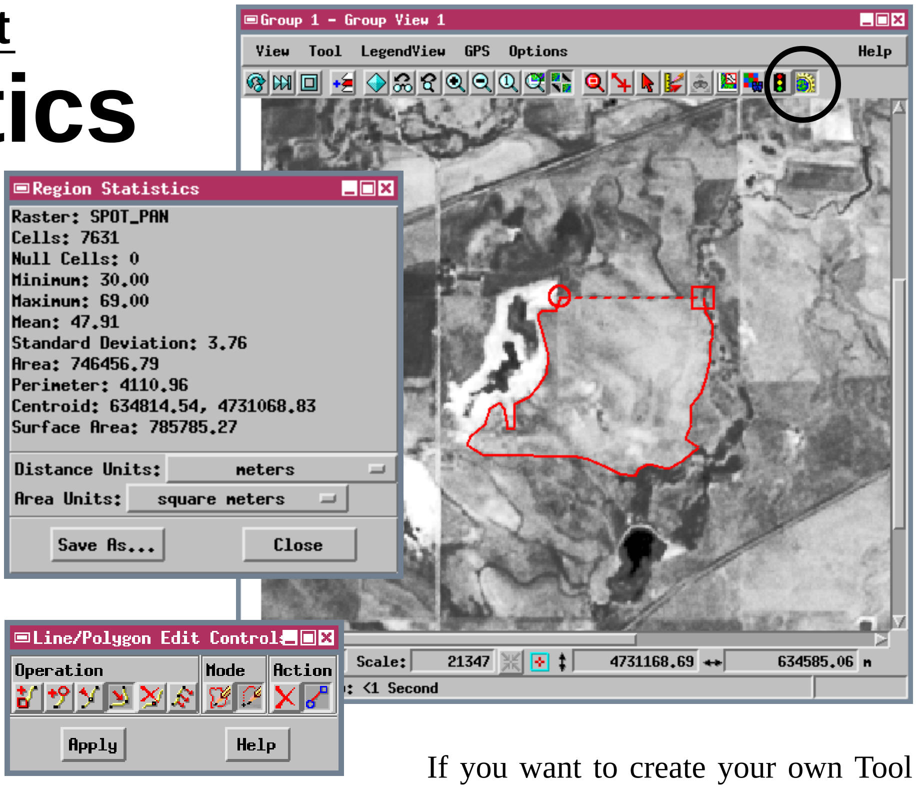
Sample SML Tool Script

Area Statistics

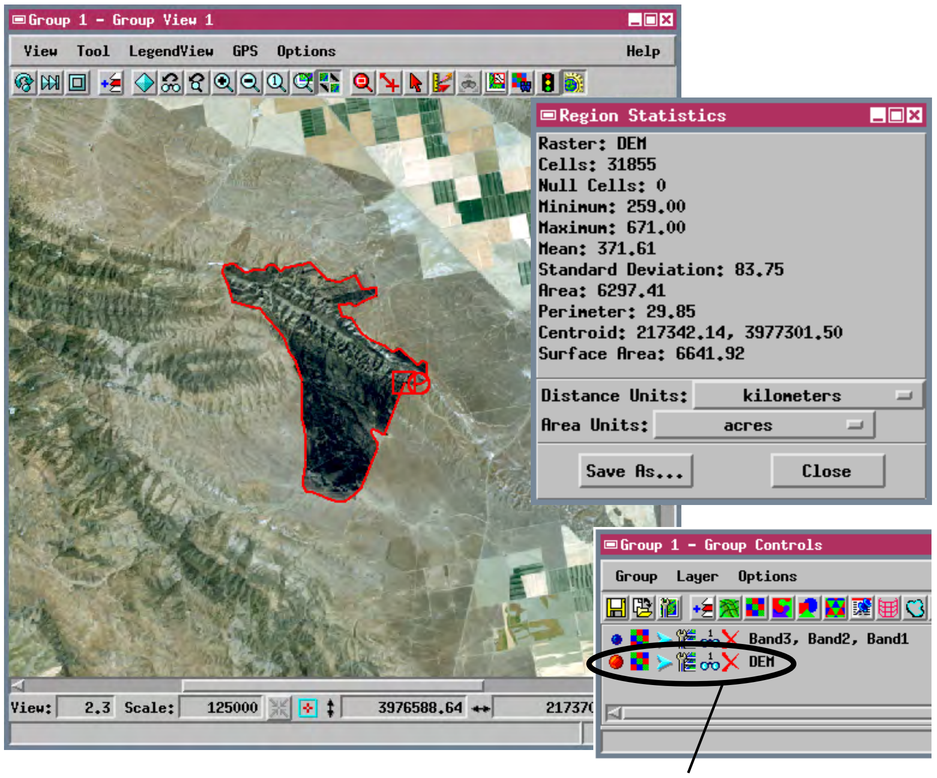
The Area Statistics script is one example of many possible applications you can create using an SML Tool Script. This sample script lets you draw a polygon in the View window and get a listing of computed raster statistics for the defined area. The Tool Script, which is shown on the other side of this page, automatically provides the graphic tool for drawing the polygon and the Region Statistics window to list the results. You could customize this script by adding other statistics computations. You can also use it as a model for building your own script that draws a polygon and uses it for some operation on one or more layers.



```
Raster: SPOT_PAN
Cells: 7631
Null Cells: 0
Minimum: 30.00
Maximum: 69.00
Mean: 47.91
Standard Deviation: 3.76
Area: 746456.79
Perimeter: 4110.96
Centroid: 634814.54, 4731068.83
Surface Area: 785785.27
```

Distance Units: meters
Area Units: square meters

If you want to create your own Tool Script, you don't have to start completely from scratch. When you create a new script you are provided with a script "skeleton": a series of commented lines that include a number of predefined functions that will be called (if used in the script) when the appropriate tool action or event occurs (as explained in the comments preceding each function). To use a function, uncomment the lines containing the start and end of the function definition, and add code in between to define what you want the function to do.

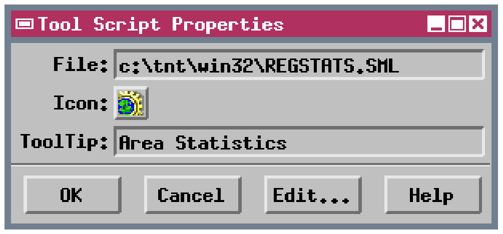


```
Raster: DEM
Cells: 31855
Null Cells: 0
Minimum: 259.00
Maximum: 671.00
Mean: 371.61
Standard Deviation: 83.75
Area: 6297.41
Perimeter: 29.85
Centroid: 217342.14, 3977301.50
Surface Area: 6641.92
```

Distance Units: kilometers
Area Units: acres

The Area Statistics Tool Script operates on the raster layer that is currently active. In the example above the DEM raster layer is active, but the polygon outlining the burned area is drawn on an overlying RGB raster layer that shows a natural-color satellite image. The resulting statistics include the minimum, maximum, and mean elevations for the burned area, which might help establish replanting schedules. When you are displaying multiple raster layers, remember to make the target raster the active layer before computing the statistics for the polygon. Statistics can be computed for any type of grayscale or binary raster, but not for RGB raster layers.

You can choose any icon (or design your own) to launch the Tool Script and provide text for its ToolTip.



Tool Script Properties

File: c:\tnt\win32\REGSTATS.SML
Icon:
ToolTip: Area Statistics

Development of this and other sample Tool Scripts continues at MicroImages. Check the MicroImages web site for an updated version incorporating additional features.

Macro and Tool Scripts can be created using SML in any TNTmips process that uses a View window (Options / Customize from the View window menu bar). These scripts are then available from an icon, which you select or design, on the toolbar. Sample scripts have been prepared to illustrate how you might use these features, which are available only in TNTmips 6.4 or later, to assist with specific tasks you perform on a regular basis. If possible, the full script is printed below for your quick perusal. When a script is too long to fit on one page, key sections are reproduced below. All sample Tool and Macro Scripts illustrated can be found in their entirety on your TNT products CD-ROM in the folder in which you installed TNTmips 6.4. These scripts, among others, can be downloaded from the SML script exchange at www.microimages.com/sml/ftpsmllink/TNT_Products_V6.4_CD.

Script for Area Statistics (regstats.sml)

```
proc cbRedraw() {
   local numeric larea, lperimeter, lsurface;

   larea = areaScale * area;
```

scales the values to specified units

```
   lperimeter = distScale * perimeter;
   lsurface = areaScale * surface;
   if (gc == 0) return;
   ActivateGC(gc);
```

redraws Region Statistics window when new statistics are computed

```
   SetColorName(“gray75”);
   FillRect(0, 0, da.width, da.height);
   SetColorName(“black”);
   if (cells > 0) {DrawInterfaceText(sprintf(“Raster: %s\nCells:
      %d\nNull Cells: %d\nMinimum: %.2f\nMaximum: %.2f\nMean:
      %.2f\nStandard Deviation: %.2f\nArea: %.2f\nPerimeter:
      %.2f\nCentroid: %.2f, %.2f\nSurface Area: %.2f”,
      rasterName$, count, cells - count, min, max, mean, stdDev, larea,
      lperimeter, centroid.x, centroid.y, lsurface), 0, 10);
      }
   else DrawInterfaceText(sprintf(“Raster: %s\nCells:\nNullCells:
      \nMinimum:\nMaximum:\nMean:\nStandardDeviation:\nArea:\nPerimeter:
      \nCentroid:\nSurface Area:”, rasterName$), 0, 10);
   }
```

main tool procedure, called within OnInitialize function

```
proc cbToolApply(class RegionTool tool) {
   if (checkLayer()) {
      local numeric sum, sumsqr, xscale, yscale, zscale;
      local numeric current, right, down, downright;
      local region MyRgn;
      local class StatusHandle status;
      local class StatusContext context;
      cells = 0; min = 0; max = 0; mean = 0; stdDev = 0; sum = 0;
      sumsqr = 0; count = 0; surface = 0; area = 0; perimeter = 0;
      centroid.x = 0; centroid.y = 0; current = 0; right = 0; down = 0;
      downright = 0;
      xscale = ColScale(targetRaster);
      yscale = LinScale(targetRaster);
      zscale = Group.ActiveLayer.zscale;
```

defines local variables for statistics calculation

```
      MyRgn = tool.Region;
      MyRgn = RegionTrans(MyRgn, ViewGetTransLayerToScreen(View,
            rasterLayer, 1));
      MyRgn = RegionTrans(MyRgn, ViewGetTransLayerToView(View,
            rasterLayer));
      MyRgn = RegionTrans(MyRgn, ViewGetTransMapToView(View,
            rasterLayer.Projection, 1));

      context = StatusContextCreate(status);
      StatusSetMessage(context, “Computing values...”);
```

creates status dialog

```
      foreach targetRaster[lin, col] in MyRgn {
         if (!IsNull(targetRaster)) {
            if (count == 0) {
               max = targetRaster;
               min = targetRaster;
               }
            else if (targetRaster > max) {
               max = targetRaster;
               }
            else if (targetRaster < min) {
               min = targetRaster;
               }
            sum += targetRaster;
            sumsqr += sqr(targetRaster);
            count += 1;
            }
```

computes statistics for the polygon

```
         if (!IsNull(targetRaster))
            current = targetRaster;
         if (!IsNull(targetRaster[lin,col+1]))
            right = targetRaster[lin,col+1];
         if (!IsNull(targetRaster[lin+1,col]))
            down = targetRaster[lin+1,col];
         if (!IsNull(targetRaster[lin+1,col+1]))
            downright = targetRaster[lin+1,col+1];
         surface += .5*sqrt(sqr(yscale*current*zscale-yscale*
            right*zscale)+sqr(xscale*current*zscale-xscale*down*zscale)
            +sqr(xscale*yscale));
         surface += .5*sqrt(sqr(yscale*downright*zscale-
               yscale*right*zscale)+sqr(xscale*downright*zscale-
               xscale*down*zscale)+sqr(xscale*yscale));
         cells += 1;
         }
      CloseRaster(targetRaster);
```

estimates surface area by adding areas of triangles created by current cell and cell below, cell to right, and cell to lower right.

```
      if (count > 1) {
         mean = sum / count;
         stdDev = sqrt((sumsqr - sqr(sum) / count) / (count - 1));
         area = MyRgn.$Data.GetArea();
         perimeter = MyRgn.$Data.GetPerimeter();
         centroid = MyRgn.$Data.GetCentroid();
         }
      cbRedraw();
```

avoids division by zero when computing mean and standard deviation.

```
      StatusContextDestroy(context);
      StatusDialogDestroy(status);
      } # end of cbToolApply
   }
```

destroys the status dialog when computations are complete

```
func OnInitialize () {
   form = CreateFormDialog(“Region Statistics”);
   form.marginHeight = 2;
   form.marginWidth = 2;
   WidgetAddCallback(form.Shell.PopdownCallback, cbClose);
```

is called the first time the tool is activated; creates the graphic tool and dialog

```
   da = CreateDrawingArea(form, 173, 301);
   da.topWidget = form;
   da.leftWidget = form;
   da.rightWidget = form;
   WidgetAddCallback(da.ExposeCallback, cbRedraw);
```

creates drawing area for dialog window

```
   line1 = CreateHorizontalSeparator(form);
   line1.topWidget = da;
   line1.leftWidget = form;
   line1.rightWidget = form;
   line1.bottomOffset = 2;
```

creates separator between statistics and menus

```
   distMenu = CreateUnitOptionMenu(form, “distance_units_c”,cbDistUnits,
      2, 0);
   distMenu.topWidget = line1;
   distMenu.leftWidget = form;
```

creates menu for selecting units

```
   areaMenu = CreateUnitOptionMenu(form, “area_units_c”, cbAreaUnits,
      1, 0);
   areaMenu.topWidget = distMenu;
   areaMenu.leftWidget = form;

   line2 = CreateHorizontalSeparator(form);
   line2.topWidget = areaMenu;
   line2.leftWidget = form;
   line2.rightWidget = form;
   line2.topOffset = 2;
```

creates separator between menus and buttons

```
   saveButton = CreatePushButtonItem(“Save As...”, cbSave);

   closeButton = CreatePushButtonItem(“Close”, cbClose);
```

creates buttons

```
   buttonRow = CreateButtonRow(form, saveButton, closeButton);
   buttonRow.topWidget = line2;
   buttonRow.leftWidget = form;
   buttonRow.rightWidget = form;
   buttonRow.bottomWidget = form;
```

creates button row

```
   tool = ViewCreatePolygonTool(View, “”, “”, “”);
   ToolAddCallback(tool.ActivateCallback, cbToolApply);
   }  # end of OnInitialize
```

creates standard polygon drawing tool

```
func OnDestroy () {
   tool.Managed = 0;
   DestroyGC(gc);
   DestroyWidget(form);
   }  # end of OnDestroy
```

destroys the tool when necessary