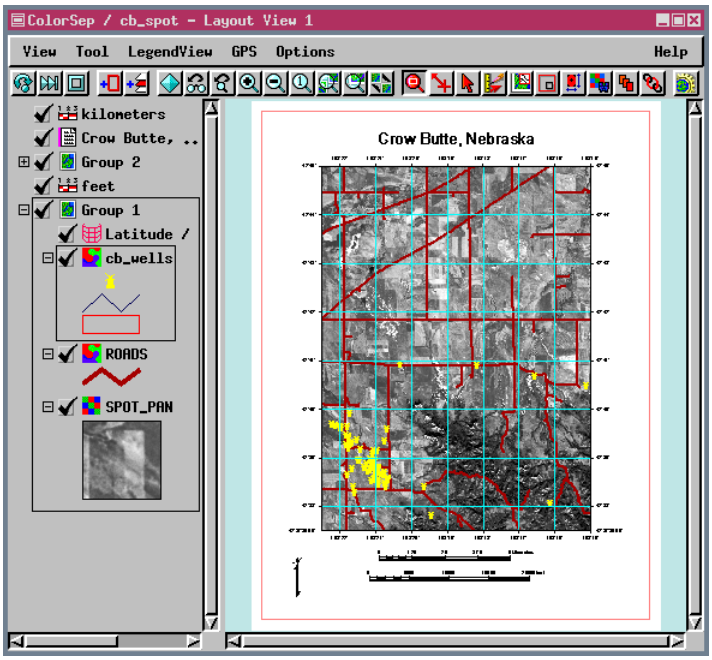
Sample SML Macro Script

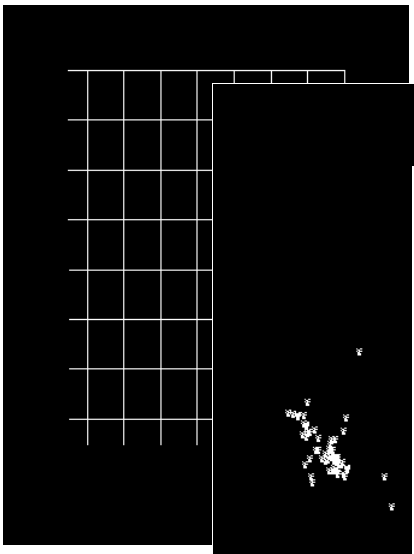
Making Color Separates for Printing



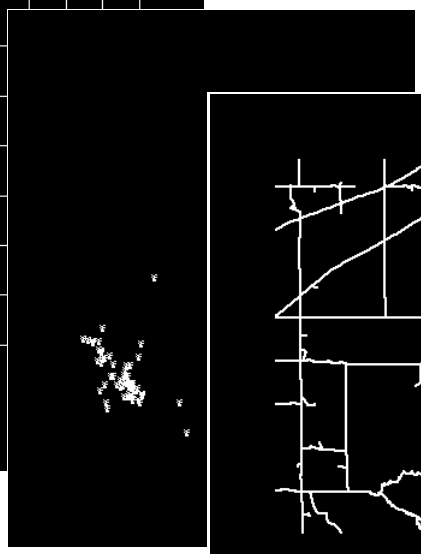
High quality cartographic products printed in volume on a press may contain features that are not screened. When color separates are used for image maps, generally only the image is screened. These features (screened or not) use a separate printing plate and press pass for each required special ink color. The ink is often specially mixed by the printer to a pantone number or other color calibration value. If a continuous tone image is added to this map, it is screened, but white areas must be left for each of the solid color ink passes.

To demonstrate the flexibility of SML, a sample Macro Script that creates the color separates needed to produce plates for this kind of volume printing has been developed. The script also produces a grayscale image suitable for screening with the color overlay areas masked out, which allows the overlay colors to be printed at the specified color without being muddied by underlying black.

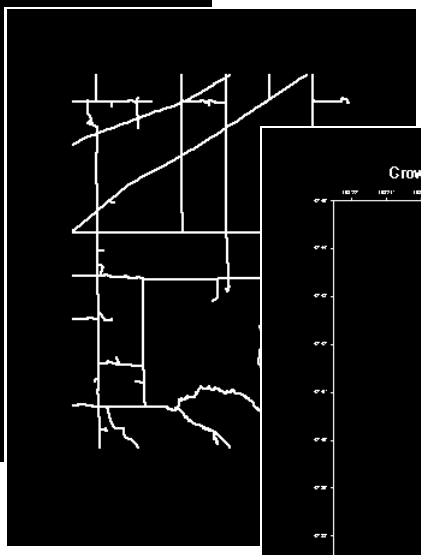
The sample script was written to handle four colors in the printing process: black, cyan, yellow, and red. These colors are used for the overlays on a grayscale image. The script creates five output rasters with preassigned names. Four of these rasters, which represent each of the color overlays, are binary TIFF files. The image raster is 8-bit grayscale TIFF with masked areas for the cyan, yellow, and red overlays. The black overlay requires no masking as it is printing black over black or gray. These TIFF images can then be used as input to digital based printing systems. The script can be easily modified to satisfy your project and printer's requirements.



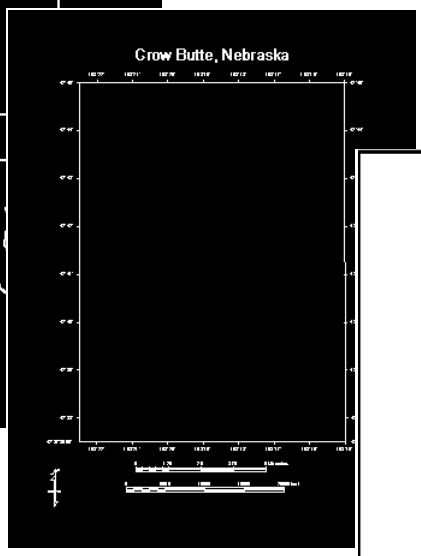
cyan



yellow

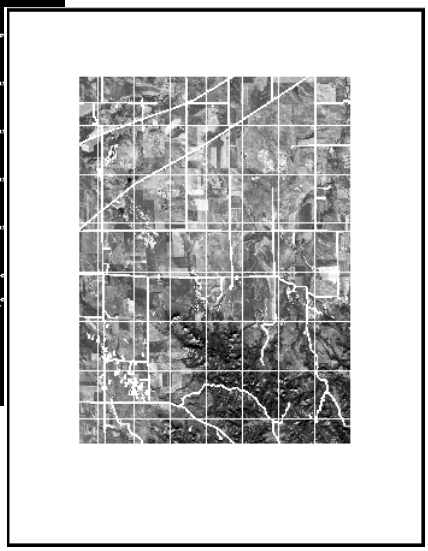


red



black

In the SML script overlays are initially rendered to a 24-bit composite color raster and then the pixels of the specified four colors are used to create binary images. The example used is simplistic in that each overlay contains only one of the four colors. Any or all of the allowed colors can be in each overlay and will be assigned to the appropriate binary raster during color separation. There is a check for pixels that are not of the appropriate colors during the color separation process. If such unknown color pixels are found, you receive a warning that provides the number of pixels of unknown color and asks if you want to proceed. These pixels are not carried through the color separation process. The script also provides a variable that will retain the work files in RVC format, such as the 24-bit composite color raster object with all overlays of the color separates before export to TIFF, for visual review testing purposes. A portion of the script is provided for reference on the back of this page.



grayscale with masks applied

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. The sample Macro Script illustrated can be downloaded from the SML script exchange at www.microimages.com/sml/ftpsmllink/TNT_Products_V6.8_CD.

Partial Script for Making Color Separates from a Layout
(printsep.sml)

```
func rgbValue (r, g, b) {
        return (((b * 256) + g) * 256 + r);
        }
```

determines RGB raster value from separate R, G, B.

```
func cleanup () {
        CloseRaster(ImageRaster);
        CloseRaster(OverlayRaster);
        if (!KeepWorkFiles) {
                imagefilename.Delete();
                overlayfilename.Delete();
                }
        statusdialog.Destroy();
        }
```

cleans up temporary files

```
func ExportBinary (inkcolor) {
        msg$ = "Exporting binary TIFF for " + inkname$;
        statuscontext.TextUpdate(msg$,2);
        if (KeepWorkFiles) {
                binaryfilename = grayscalefilename;
                }
        else {
                binaryfilename = CreateTempFileName();
                }
        binaryrastname$ = "B_" + inkname$;
        CreateRasterBinaryMask(OverlayRaster,binaryfilename,
              binaryrastname$,inkcolor);
        targetpath = targetdir$;
        targetpath.Append(inkname$);
        targetpath.Make();
        targetpath.Append(filename$);
        ExportRaster(tiffexp,targetpath,binaryfilename,binaryrastname$);
        if (!KeepWorkFiles) {
                binaryfilename.Delete();
                }
        }
```

exports binary TIFF for specified ink color

```
targetdir$ = GetDirectory("c:/temp","Select destination folder for TIFF
      separates:");
if (targetdir$ == "") Exit();
```

prompts user for target directory folder

```
imageres = PopupNum("Image resolution in DPI?",300,50,1200,0);
if (imageres < 0) Exit();
overlayres = PopupNum("Overlay resolution in DPI?",1200,imageres,
      2400,0);
if (overlayres < 0) Exit();

statusdialog.Create(View.Form);
statuscontext = statusdialog.CreateContext();
statuscontext.Message = "Rendering Color Separations";

resratio = overlayres / imageres;

imagecellsize = Layout.MapScale / imageres * .0254;
overlaycellsize = imagecellsize * resratio;
```

determines image and overlay resolutions.

creates progress dialog

resolution ratio between overlay and image

computes raster cell sizes in meters

```
group = Layout.Firstgroup;
while (group != 0) {
        layer = group.FirstLayer;
        while (layer != 0) {
                if (layer.IsVisibleInView(hardcopyviewnum)) {
                        isimage = (layer.Type == "Raster");
                        layer.SetVisibleInView(imageviewnum,
                              isimage);
                        layer.SetVisibleInView(overlayviewnum,
                              !isimage);
                        }
                else {
                        layer.SetVisibleInView(imageviewnum,0);
                        layer.SetVisibleInView(overlayviewnum,0);
                        }
                layer = layer.NextLayer;
                }
        group = group.NextGroup;
        }
```

sets layer visibility based on whether image and visibility in 'hardcopy' view

```
if (KeepWorkFiles) {
        imagefilename = targetdir$;
        imagefilename.Append("image.rvc");
        imagefilename.Delete();
        }
else {
        imagefilename = CreateTempFileName();
        }
statuscontext.TextUpdate("Rendering images",2);
errcode = Layout.RenderToRaster(imagefilename,"Image",imageviewnum,
      imagecellsize);
if (errcode < 0) {
        PopupError(errcode);
        Exit();
        }
```

renders image raster

```
OpenRaster(ImageRaster,imagefilename,"Image");
imagenumcols = NumCols(ImageRaster);
imagenumlins = NumLins(ImageRaster);
```

opens image raster and determines size

```
overlaynumcols = imagenumcols * resratio;
overlaynumlins = imagenumlins * resratio;
```

makes overlay raster exact multiple of image raster in size

```
if (KeepWorkFiles) {
        overlayfilename = targetdir$;
        overlayfilename.Append("overlay.rvc");
        overlayfilename.Delete();
        }
else {
        overlayfilename = CreateTempFileName();
        }
statuscontext.TextUpdate("Rendering overlays",2);
```

renders the overlay raster