

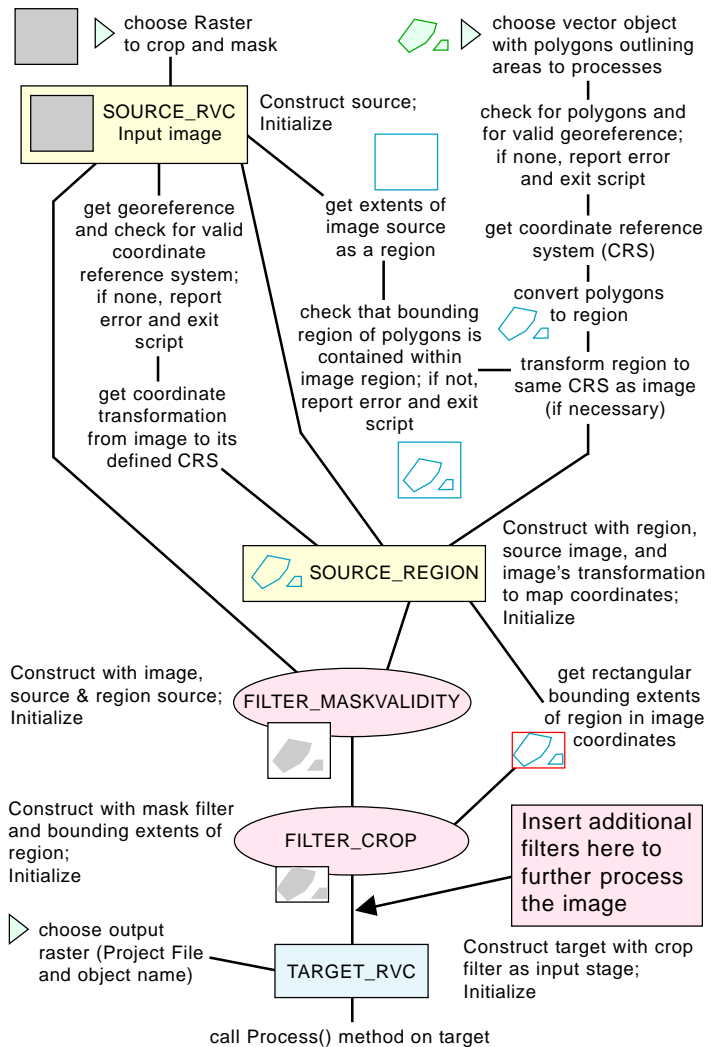
Using Regions in a Pipeline

A *region* is a spatial object that is used to outline simple or complex areas of interest. TNT geospatial scripts that implement an image processing pipeline can use region objects to mask an image and/or to crop an image to the bounding extents of the region. The region used for such operations can be input to the script, obtained from another image in the pipeline, or computed from a geometric object (such as vector polygons). The illustration to the right diagrams a sample script (PipelineCropAndMaskFromRegion.sml, excerpted on the reverse) that computes a region from polygons in a vector object you select and uses this region to mask and crop an image you select.

To use a region in the pipeline, the script constructs an instance of class `IMAGE_PIPELINE_SOURCE_REGION` using the region and (as reference) the input image source. The resulting region source has the dimensions of the source image and is registered to it by means of its georeference. (If the vector object used to create the region has a different coordinate reference system [CRS] than the reference image, the script reprojects the derived region to the image CRS before creating the region source.) The `SOURCE_REGION` is used as input to the *mask validity* pipeline filter, which masks out the portion of the input image outside of the region's closed outlines by creating a null mask for the image. The `SOURCE_REGION` class also has a method to obtain the bounding rectangular extents of the region (i.e., the bounding extents of the aggregate of all closed outlines in the region) in image coordinates. The script uses these rectangular bounding extents as input to the *crop* pipeline filter to crop the image.

All pipeline stages are also provided with a class method to obtain the extents of that stage as a region object. This method is used in the sample script to obtain a region from the image source and test whether the region computed from the input vector polygons is contained within the image extents. If it is not, the script reports an error and exits.

This sample script could be modified easily to add filters to perform additional processing to the image, such as changing its CRS, applying a sharpening filter, or changing cell size. The sample script entitled PipelineNDVIFromTIFF_CropAndMaskFromRegion.sml shows how region mask and crop operations can be grafted onto a more complex image processing pipeline.



Annotated diagram of pipeline to mask and crop an image using a region created from vector polygons (see PipelineCropAndMaskFromRegion.sml on reverse).

A sample source image is shown to the left (color-composite with UTM coordinate reference system) with overlay of vector polygons (transparent yellow fill) with a geographic CRS. The sample script converts these polygons to a region, transforms the region to the image CRS, and creates a `SOURCE_REGION` for use in a pipeline to mask and crop the image. The resulting target image with UTM CRS is shown below with a black border added to indicate its extents. White areas of the target in this illustration are null-masked.



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.

Excerpts of Pipeline Script to Crop and Mask an Image Using a Region from Vector Polygons (PipelineCropAndMaskFromRegion.sml)

```
class RVC_OBITEM rastInObjItem;
DlgGetObject("Choose input raster:", "Raster", rastInObjItem, "ExistingOnly");
```

CHOOSE INPUT RASTER

```
class IMAGE_PIPELINE_SOURCE_RVC sourceImg(rastInObjItem);
err = sourceImg.Initialize();
```

PIPELINE SOURCE FOR IMAGE

```
if (err < 0)
  ReportError(_context.CurrentLineNum, err);
else {
  numeric numLines, numCols;
  numLines = sourceImg.GetTotalRows();
  numCols = sourceImg.GetTotalColumns();
  print("Source image initialized.");
  printf("Image size = %d lines, %d columns\n", numLines, numCols); }

```

Get georeference and coordinate reference system from the image

```
class IMAGE_PIPELINE_GEOREFERENCE georefImg;
georefImg = sourceImg.GetGeoreference();
```

```
class SR_COORDREFSYS imgCRS = georefImg.GetCRS();
```

```
if (!imgCRS.IsDefined()) {
  print("Image is not georeferenced; exiting now.");
  Exit(); }
else if (imgCRS.IsLocal()) {
  print("Image has local engineering georeference; exiting now.");
  Exit(); }
else print("Image coordinate reference system = %s\n", imgCRS.Name );
```

Get coordinate transformation from image to its defined CRS

```
class TRANS2D_MAPGEN imgToCRS = georefImg.GetTransGen();
```

CHOOSE VECTOR OBJECT WITH POLYGONS
TO INDICATE AREAS TO PROCESS

```
class RVC_VECTOR polyVect; class RVC_OBITEM vectObjItem;
class RVC_GEOREFERENCE vectGeoref; class SR_COORDREFSYS vectCRS;
DlgGetObject("Choose vector object with polygons outlining areas to process:",
  "Vector", vectObjItem, "ExistingOnly");
polyVect.Open(vectObjItem, "Read");
```

check for polygons

```
if (polyVect.$Info.NumPolys == 0) {
  print("Vector object contains no polygons; exiting now.");
  Exit(); }
```

```
err = polyVect.GetDefaultGeoref(vectGeoref);
```

check for georeference

```
if (err < 0) {
  ReportError(_context.CurrentLineNum, err);
  print("Polygon vector must be georeferenced and is not; exiting now.");
  Exit(); }
```

```
vectCRS = vectGeoref.GetCoordRefSys();
printf("Vector coordinate reference system = %s\n", vectCRS.Name);
```

CONVERT VECTOR POLYGONS TO A REGION; if vector has
different coordinate reference system from image, reproject region

```
array numeric polynums[0];
```

array of non-island polygon numbers

```
for i = 1 to polyVect.$Info.NumPolys {
  if (polyVect.poly[i].Internal.Inside == 0) {
    ResizeArrayPreserve(polynums, ++j);
    polynums[j] = polyVect.poly[i].Internal.ElemNum; }
}
```

only use polygons that are
not islands to make region

```
class REGION2D RegFromVect = ConvertVectorPolysToRegion(polyVect,
  GetLastUsedGeorefObject(polyVect), polynums, j);
```

```
if (vectCRS.Name <> imgCRS.Name) {
  printf("Converting region to image CRS: %s\n", imgCRS.Name);
  RegFromVect.ConvertTo(imgCRS); }
```

GET EXTENTS OF THE IMAGE SOURCE AS A REGION

```
class REGION2D imgReg;
err = sourceImg.ComputeGeoreferenceRegion(imgReg);
if (err < 0)
  ReportError(_context.CurrentLineNum, err);
```

CHECK THAT REGION FROM VECTOR IS
CONTAINED WITHIN IMAGE REGION

```
if (imgReg.TestRegion(RegFromVect, "FullInside")) {
  print("Image region contains vector extents."); }
else {
  print("Input vector is not contained within the source image. Exiting now.");
  Exit(); }
```

PIPELINE SOURCE FOR REGION CREATED FROM THE
VECTOR POLYGONS; construct using TRANS2D_MAPGEN
with coordinate transformation from source image to its CRS.
Use source image as a reference so the "image dimensions"
of the region source match, enabling the region source to be
used as a mask in the MASKVALIDITY filter

```
class IMAGE_PIPELINE_SOURCE_REGION sourceReg(RegFromVect,
  imgToCRS, sourceImg);
err = sourceReg.Initialize();
if (err < 0) ReportError(_context.CurrentLineNum, err);
else {
  print("Initialized region source.");
  printf("Size of region = %d lines, %d columns\n", sourceReg.GetTotalRows(),
    sourceReg.GetTotalColumns()); }
```

Get the rectangular extents of the region in image coordinates to
use to crop the source image. These are the extents of the
region used to create the region source, not those of the image
used as its reference.

```
class RECT regExtents;
regExtents = sourceReg.GetRegionExtents();
printf("Minimum values of cropping rectangle from polygons: x = %.2f, y = %.2f\n",
  regExtents.x1, regExtents.y1);
printf("Maximum values of cropping rectangle from polygons: x = %.2f, y = %.2f\n",
  regExtents.x2, regExtents.y2);
```

PIPELINE FILTER TO MASK PORTIONS OF THE IMAGE;
mask area outside the region created from the vector polygons

```
class IMAGE_PIPELINE_FILTER_MASKVALIDITY filterMask(sourceImg,
  sourceReg);
err = filterMask.Initialize();
if (err < 0) ReportError(_context.CurrentLineNum, err);
else print("Initialized image mask filter.");
```

PIPELINE FILTER TO CROP the image to the extents of the region

```
class IMAGE_PIPELINE_FILTER_CROP filterCrop(filterMask, regExtents);
err = filterCrop.Initialize();
if (err < 0) ReportError(_context.CurrentLineNum, err);
else print("Initialized image crop filter.");
```

PIPELINE TARGET: set up the target for the pipeline

```
class IMAGE_PIPELINE_TARGET_RVC target(filterCrop, rastOutObjItem);
err = target.Initialize();
if (err < 0) ReportError(_context.CurrentLineNum, err);
else print("Initialized target.");
```

```
target.Process();
```

EXECUTE pipeline process

```
print("Done.");
```