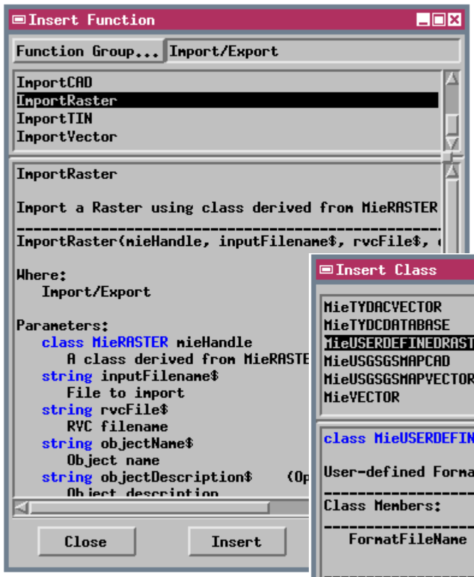
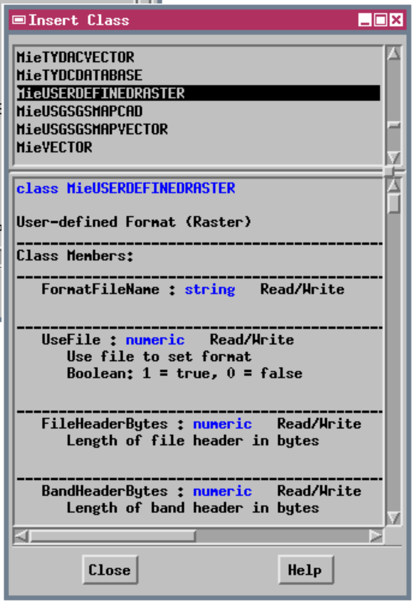
Automate Batch Imports using SML

The TNT products are widely known for the great number and variety of data formats supported in the interactive Import / Export process. But if you have a large number of files in a particular format to process, or an import is only the first step in a multi-step process sequence, using an interactive dialog may not be the most efficient way to handle the data. The Spatial Manipulation Language (SML) provides all the tools you need to write custom scripts to automate the import or export of spatial data in almost any fomat.



```
Insert Function
Function Group...  Import/Export

ImportCAD
ImportRaster
ImportTIN
ImportVector

ImportRaster

Import a Raster using class derived from MieRASTER
---------------------------------------------------
ImportRaster(mieHandle, inputFilename$, rvcFile$, 

Where:
   Import/Export

Parameters:
   class MieRASTER mieHandle
      A class derived from MieRASTE
   string inputFilename$
      File to import
   string rvcFile$
      RVC filename
   string objectName$
      Object name
   string objectDescription$     (Op
      Object description
```



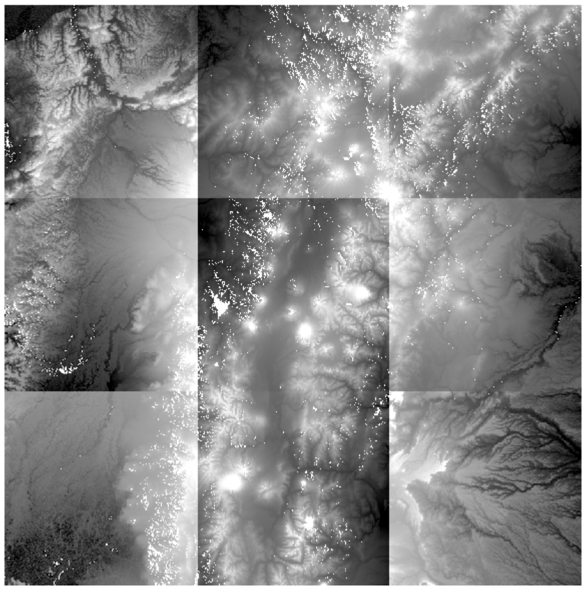
```
Insert Class

MieTYDACVECTOR
MieTYDCDATABASE
MieUSERDEFINEDRASTER
MieUSGSGSMAPCAD
MieUSGSGSMAPVECTOR
MieVECTOR

class MieUSERDEFINEDRASTER

User-defined Format (Raster)
----------------------------------------------
Class Members:
----------------------------------------------
   FormatFileName : string   Read/Write

----------------------------------------------
   UseFile : numeric   Read/Write
      Use file to set format
      Boolean: 1 = true, 0 = false

----------------------------------------------
   FileHeaderBytes : numeric   Read/Write
      Length of file header in bytes

----------------------------------------------
   BandHeaderBytes : numeric   Read/Write
      Length of band header in bytes
```

Use the SML Import / Export functions and the Mie format classes to write an SML script to import or export your spatial data.

Each of the file formats supported for Import / Export has a corresponding SML class (beginning with letters "Mie", for "MicroImages Import/Export") with class members that you can use to specify the necessary import or export settings specific to that format. Once you have declared an instance of the relevant Mie... class and set the desired values for its class members, the script should call the function in the Import / Export function group that is appropriate for your data type and processing direction, such as ImportRaster(). Your script should pass the Mie class variable as a parameter to this function, allowing the generic import or export function to process the specified data format with your specified settings.

As an example of a script that automates batch import, MicroImages has created the sample script IMPORT_SRTM.SML, which is shown on the other side of this page. This script is designed to import preliminary elevation grids provided by NASA's Shuttle Radar Topography Mission (SRTM; see box below). This elevation data is distributed in raw binary files with 16-bit signed integer elevation values. Each .hgt file is a grid with 1201 lines and 1201 columns sampled at three arc-seconds of latitude and longitude, covering one square degree. To import these files, the script uses the MieUSERDEFINEDRASTER class, whose members allow you to specify the data type, byte order, number of lines and columns, and other parameters for a custom raster import.

The SML script is designed to import multiple elevation files in a single directory. The SRTM height file names incorporate the latitude and longitude of the lower left (southwest) corner of the one-degree grid, so this information can be used to georeference the imported grids. For each file, the script parses the file name to determine the reference latitude and longitude, imports the file, then creates a georeference subobject with corner control points. MicroImages has used this script to import all of the one-degree tiles covering South America, a total of 1813 files representing 4.9 GB of data.



The illustration above shows a 3 x 3 block of the imported SRTM elevation grids for South America, covering part of Ecuador. Each elevation raster covers one degree of latitude and longitude, and each is displayed with contrast stretched to its own elevation range, so gray tones do not match at the raster boundaries. Scattered irregular white areas are null (no data) areas corresponding to radar shadows.

About SRTM DATA

The SRTM mission, flown on the Space Shuttle Endeavour in February 2000, used an interferometric imaging radar system with dual radar antennas to collect topographic data over nearly 80% of Earth's land surfaces. The preliminary digital elevation grids currently being distributed to researchers and the public are not fully processed: they contain numerous areas without data, water bodies that are not flat, and ill-defined coastlines. Corrected data will be distributed at a later date. For further information about about SRTM data see: http://www.jpl.nasa.gov/srtm/. Data download is available from: ftp://edcsgs9.cr.usgs.gov/pub/data/srtm/.

Sample scripts have been prepared to illustrate how you might use the features of TNTmips' Spatial Manipulation Language (SML). 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 script illustrated can be downloaded from the SML script exchange at www.microimages.com/freestuf/smlscripts.htm.

Script for batch import of SRTM elevation grids (import_srtm.sml)

This script imports and sets the georeference for the 'headerless' *.hgt SRTM-3 files.

```
clear();

class FILEPATH filepath;
class STRINGLIST filenames;
class MieUSERDEFINEDRASTER srtm;
```

get the latitude value from the *.hgt filename

```
func getLatitude( hgtfile$ )    {
      local numeric latvalue = StrToNum( GetToken( hgtfile$, "NSEW.hgt", 1, 1 ) );
      string hemisphere = GetToken( hgtfile$, "EW.hgt0123456789", 1, 1 );

      if ( hemisphere == "N" )  {
            return latvalue;
      }
      else if ( hemisphere == "S" )  {
            return -latvalue;
      }
      else  {
            print( "Error parsing filename for latitude" );
            print( "Bad file was: " + hgtfile$ );
            return -9999;
      }
}
```

if northern hemisphere

if southern hemisphere

get the longitude value from the *.hgt filename

```
func getLongitude( hgtfile$ ) {
      local numeric longvalue = StrToNum(GetToken(hgtfile$, "NSEW.hgt",2,1) );
      string hemisphere = GetToken( hgtfile$, "NS.hgt0123456789", 1, 1 );

      if ( hemisphere == "E" )  {
            return longvalue;
      }
      else if ( hemisphere == "W" )       {
            return -longvalue;
      }
      else       {
            print( "Error parsing filename for longitude" );
            print( "Bad file was: " + hgtfile$ );
            return -9999;
      }
}
```

if eastern hemisphere

if western hemisphere

set parameters for USERDEFINEDRASTER import

```
proc setImportParameters(path$)       {
      srtm.NumLins = 1201;
      srtm.NumCols = 1201;
      srtm.DataType = "16-bit signed";
      srtm.ByteOrder = "High-Low";
      srtm.UseFile = 0;
      srtm.DoPyramid = 1;
      srtm.DoCompress = 1;
}
```

get the SRTM data directory

```
string defaultpath$ = _context.ScriptDir;
filepath.SetName( GetDirectory( defaultpath$,
            "Please select the directory containing the SRTM files for import" ) );
setImportParameters( filepath.GetPath() );
print( filepath.GetPath() );
filenames = filepath.GetFileList( "*.hgt" );

numeric filecount = filenames.GetNumItems();
print( filecount, "files found" );
```

get the output file name

```
string outputfile$ = GetOutputFileName(filepath.GetName(),
            "Enter a file name for the imported srtm raster rvc", ".rvc" );

numeric lat, long;
numeric i;
```

```
for i = 0 to filecount-1      {
      print( "Now importing: " + filenames.GetString( i ) + " raster number: " +
            NumToStr( i + 1 ) );
      lat = getLatitude( filenames.GetString( i ) );
      long = getLongitude( filenames.GetString( i ) );

      if ( lat != -9999 && long != -9999 )          {
            string inputfile$ = filepath.GetPath() + "\\" + filenames.GetString( i );
            string objname$ = FileNameGetName( filenames.GetString( i ) );

            ImportRaster( srtm, inputfile$, outputfile$, objname$,
                        "file imported from: " + filenames.GetString( i ) );
```

import raster

'open' the raster so that it can be georeferenced

```
            Raster importedRaster;
            OpenRaster( importedRaster, outputfile$, objname$ );

            SetNull( importedRaster, -32768 );

            numeric xmin, ymin, xmax, ymax;
            xmin = 0.5;             # left edge
            xmax = 1200.5;          # right edge
            ymin = 0.5;             # top edge
            ymax = 1200.5;          # bottom edge
```

set the raster extents (in object coordinates) based on centers of corner cells

```
            array numeric sourceX[3];
            array numeric sourceY[3];
            array numeric destX[3];
            array numeric destY[3];

            sourceX[1] = xmin;
            sourceY[1] = ymin;
            sourceX[2] = xmin;
            sourceY[2] = ymax;
            sourceX[3] = xmax;
            sourceY[3] = ymin;

            destX[1] = long;
            destY[1] = lat + 1;
            destX[2] = long;
            destY[2] = lat;
            destX[3] = long + 1;
            destY[3] = lat + 1;
```

save the object extents as the 'source' and the latitude / longitude values as 'dest' arrays to be used to create control points at the raster corners:
[1] = upper left corner,
[2] = lower left corner,
[3] = upper right corner

```
            class MAPPROJ mapproj;
            mapproj.SetSystemLatLon();
            mapproj.Datum = "World Geodetic System 1984";
```

set projection parameters

use 'source' and 'dest' arrays to create control point georeference for the object

```
            CreateControlPointGeorefDefaultAccuracy( importedRaster, mapproj, 3,
                  sourceX, sourceY, destX, destY );

            CloseRaster( importedRaster );
      }
      else       {
            print( "Check file name and structure, lat/long not computed correctly" );
      }
}

string s$ = "s";
```

take care of plural for final print statement if needed

```
if ( filecount == 1 )          {
      s$ = "";
}
```

let user know that it's done

```
print( "Done. ", filecount, "raster" + s$ + " imported." );
```