

**USDA Service Center Agencies  
Geospatial Data Management Team  
Data Management Plan For**

**Elevation - IFSAR Data**

**June 2008**

**Ken Becker, Janice Sterling and Randy Frosh**

**I. Purpose and Scope (business case)**

**A. Purpose**

These maps are created from X-band InterFerometric Synthetic Aperture Radar (IFSAR) high-resolution digital elevation maps. IFSAR operates day or night, in clear or cloudy conditions. The maps are developed by [Intermap Inc.](#), a global provider of high-resolution digital elevation map products. The product handbook, located at <http://www.intermap.com/uploads/1170106364.pdf>, contains information on the products, applications, technology, how to load the maps into various GIS software packages and product licensing agreements.

The maps are GeoTiff format 7.5-minute by 7.5-minute units, corresponding to the USGS 1:24,000 scale topographic quadrangle map series for available areas in the United States and throughout the world. Each 7.5-minute by 7.5-minute tile provides full coverage with overlap into adjacent tiles. Data for locations above 56 degrees North/South are licensed in 7.5-minute by 15-minute tiles.

There are four different geospatial datasets: Orthorectified Radar Imagery (ORI), Digital Surface Models (DSM), and Digital Terrain Models (DTM), and Correlation maps (COR). All four types are GeoTiff format. The units for DSM and DTM are meters. The units for COR and ORI are 0-255. The horizontal coordinates for all types are northings/eastings in meters. Spatial Reference Information: Universal Transverse Mercator (UTM), NAD83 horizontal datum and NAVD88 (Geoid99) vertical datum.

An ORI is a grayscale image of the earth's surface that has been corrected to remove geometrical distortions that are a normal part of the imaging process. This product looks similar to a black-and-white aerial photograph. The difference is that, instead of being made of visible light, the radar pulses the ground with "flashes" of radio waves, which then return from the ground (or whatever they strike, including buildings and trees) to the antennae to give distance and intensity measurements. The key feature of this product is that it provides a means of viewing the earth's surface in a way that accentuates features far more than is possible with aerial photography. The radar looks to the side of the aircraft and casts "shadows" that enable users to visually perceive the elevation information in the image, even if they are unfamiliar with the underlying technology.

A DSM is a topographic model of the earth's surface that can be manipulated using a computer. It is comprised of elevation measurements that are laid out on a grid. These measurements are derived from the return signals received by the two radar antennae on the aircraft. The signals bounce off the first surface they strike, making the DSM a representation of any object large enough to be resolved. This includes buildings, vegetation and roads, as well as natural terrain features. The key feature of this product is that it provides a geometrically correct reference frame over which other data layers can be draped.

A DTM is a topographic model of the bare earth that can be manipulated using a computer. A DTM has had vegetation, buildings and other cultural features digitally removed, leaving just the underlying terrain. This is achieved using Intermap proprietary software called TerrainFit, which

derives terrain elevations based on measurements of bare ground contained in the original radar data (DSM). The key feature of a DTM is that it enables users to infer those terrain characteristics that may be hidden in the DSM.

A *COR* is a correlation map co-registered with the elevation models. This map represents the degree of correlation between the two radar signals received by the IFSAR system on a pixel by pixel basis. The higher the value, the closer the two signals agreed and therefore, the more confidence that can be placed on the accuracy of that pixel. In difficult terrains (steep slopes, dense urban areas, and areas of shadow), the signal may have such low correlation that the pixel in the DSM will be left blank in the interferometric processing phase and subsequently interpolated based on surrounding pixels. In the *COR* file, pixels that have been interpolated will be set to 0, while all remaining pixels will vary in value from 1 to 255, with the brightest pixels (255) having the best correlation. There are no horizontal nor vertical accuracies associated with this data set.

## **B. Scope**

See the status maps at: <http://datagateway.nrcs.usda.gov/statusmaps.aspx> for current data availability. The dataset will continue to grow as more data is received from the contractor.

## **II. Acquisition**

### **A. Data Source**

#### **1. Producer Information**

##### **a. Name**

Intermap Technologies, Inc.

##### **b. Location of Headquarters**

Intermap Technologies, Inc.  
400 Inverness Parkway, Suite 330  
Englewood, CO 80112-5847

##### **c. Internet Address**

<http://www.intermap.com/>

#### **2. Publisher Information**

##### **a. Name**

Intermap Technologies, Inc.

##### **b. Location of Headquarters**

400 Inverness Parkway, Suite 330  
Englewood, CO 80112-5847

##### **c. Internet Address**

<http://www.intermap.com>

#### **3. Acquisition Information**

##### **a. Delivery Media**

DVDs for > 4 GB and < 20 GB

USB hard disc for > 20 GB.

- b. Download URL

[n/a](#)

- c. Projected Data Availability Schedule

As contracted

## **B. Standards Information**

### **1. Geospatial Data Standard**

- a. Standard Name and Steward Information

From Intermap product handbook, located at <http://www.intermap.com/uploads/1170106364.pdf>, “Core products are created according to tightly controlled specifications. These products only vary when the specifications are upgraded—to reflect improvements in hardware, for example.

Intermap is an ISO 9001:2000-registered company, audited on a regular basis by Underwriters Laboratories, Inc. Underwriters Laboratories is an independent company that checks to ensure we document and then follow our procedures for acquiring and processing data. We have put stringent controls in place because we know they save time and money. This makes us more competitive and ensures that you will receive the products you ordered on time. Another stipulation of being ISO-registered is that we must have a defined process for correcting problems when they occur, and take measures to ensure that the problems don’t happen again.”

- b. Standard Version

None

- c. Standard URL

See the product handbook at <http://www.intermap.com/uploads/1170106364.pdf>

### **2. Metadata Standard**

- a. Standard Name and Steward Information

FGDC Content Standards for Digital Geospatial Metadata FGDC-STD-001-1998

Data Archive Manager  
Intermap Technologies Inc.  
400 Inverness Parkway, Suite 330  
Englewood , CO 80112-5809  
*Contact\_Voice\_Telephone:* (303) 708-0955  
*Contact\_Facsimile\_Telephone:* (303) 708-0952  
*Contact\_Electronic\_Mail\_Address:* [<mailto:info@intermap.com>](mailto:info@intermap.com)

- b. Description of Metadata Captured

FGDC Content Standards for Digital Geospatial Metadata FGDC-STD-001-1998

- c. Metadata Accuracy and Completeness Assessment

In compliance with FGDC Content Standards for Digital Geospatial Metadata FGDC-STD-001-1998

### **C. Acquired Data Structure**

#### **1. Geospatial Data Format**

- a. Format (raster, vector, etc.)

Raster

- b. Format Name

TIFF is a non-proprietary format. It is a 32-bit floating-point raster format.

- c. Data Extent

See status map on the Data Gateway: <http://datagateway.nrcs.usda.gov/statusmaps.aspx>  
The dataset will continue to grow as more data is received from the contractor.

- d. Horizontal and Vertical Resolution

The ORI has a pixel resolution of 1.25 meters, while the DSM and DTM are products with 1-meter RMSE vertical accuracy, posted at 5-meter intervals.

- e. Absolute Horizontal and Vertical Accuracy

The ORI has a pixel resolution of 1.25 meters, while the DSM and DTM are products with 1-meter RMSE vertical accuracy, posted at 5-meter intervals.

- f. Nominal Scale

1:12,000

- g. Horizontal and Vertical Datum

The horizontal datum for all areas is NAD83. The vertical datum is NAVD88.

- h. Projection

UTM

- i. Coordinate Units

Meters

- j. Average Data Set Size

The combined file size for a complete dataset (ORI, DSM, DTM, and metadata) is nearly one megabyte per square kilometer. The breakdown, per 100 square kilometers, is as follows:

- DSM and DTM products are approximately 16 MB each
- Corresponding ORI is 65 MB

- k. Symbology

None

## 2. Attribute Data Format

### a. Format Name

N/A - Raster data.

### b. Database Size

N/A

## 3. Data Model

### a. Geospatial Data Structure

TIFF is a non-proprietary format. It is a 32 bit floating point grid format.

### b. Attribute Data Structure

N/A - Raster data.

### c. Database Table Definition

N/A - Raster data.

### d. Data Relationship Definition

N/A - Raster data.

### e. Data Dictionary

See product handbook at: <http://www.intermap.com/uploads/1170106364.pdf>

## **D. Policies**

### 1. Restrictions

#### a. Use Constraints

The Intermap IFSAR data is only available to the SCA and USDA agencies that have a current paid subscription.

U.S. Government End Users - The Product is a “commercial item” as that term is defined at 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 (Sept. 1995) and 48 C.F.R. 227.7202-1 throughout 227.7202-4 (June 1995), all U.S. Government End Users acquire the Products with only those rights set forth herein. Contractor/manufacturer is Intermap Technologies Incorporated, Englewood, Colorado, USA. If the Products or any Thematic Derivative Works are used in connection with the performance of any government contracts or subcontracts, You shall ensure that (i) the Products and any Thematic Derivative Works shall not constitute a deliverable under any governmental contracts or subcontracts; and (ii) in no event shall a government entity acquire any rights other than those provided in “Appendix A - INTERMAP TECHNOLOGIES INC. PROJECT END USER LICENSE AGREEMENT” – in <http://www.intermap.com/uploads/1170106364.pdf>

#### b. Access Constraints

The Intermap IFSAR data is only available to the SCA and USDA agencies that have a current paid subscription. IFSAR data can be used to produce digital and hard copy map products, but Intermap IFSAR data are not to be redistributed outside of USDA. Intermap IFSAR data will be available through the Data Gateway to those licensed USDA agencies. Intermap retains all rights to the data.

c. Certification Issues

None

2. Maintenance

a. Temporal Information

There is no temporal information because this is elevation data.

b. Average Update Cycle

As more IFSAR data is purchased from Intermap, it will become available on the Data Gateway.

**E. Acquisition Cost**

1. Cooperative Agreement

a. Description of Agreement

As more IFSAR data is purchased from Intermap, it will become available on the Data Gateway.

b. Status of Agreement

Unknown

3. Cost to Acquire Data

Unknown

**III. Integration**

**A. Value Added Process**

1. Benefit to the Service Center

The IFSAR data is organized by 7.5 minute quads. High resolution elevation data is beneficial to many SCA and USDA agencies that require high resolution elevation data. The ORI has a pixel resolution of 1.25 meters, while the DSM and DTM are Type II products with 1-meter RMSE vertical accuracy, posted at 5-meter intervals.

2. Process Model

a. Flow Diagram

b. Process Description

The copyIFSAR script must be run when the maps are received. This script (see attached):

1. processes the data and moves it into the proper zone/block utm structure
2. converts it to geoTiff with COPYRASTER
3. determines correct utm zone, based on longitude of data
4. calculates statistics, and builds pyramids
5. The following extensions should be in the proper zone/block utm structure to copy to the ifsar repository, \\v480d\qfs1\ifsar. Any other extensions will result in error messages, and the tifs will not be created as footprints when running subsequent programs.

#### IFSARCOR

n32w100h5cor.tif  
.tif  
.rrd  
.aux  
.tif.xml

#### IFSARDSM

n32w100h5dsm.tif  
.tif  
.rrd  
.aux  
.tif.xml  
.html  
.txt  
.xml

#### IFSARDTM

.tif  
.rrd  
.aux  
.tif.xml  
.html  
.txt  
.xml

#### IFSARORI

.tif  
.rrd  
.aux  
.tif.xml  
.html  
.txt  
.xml

For Creating Derivative Products for Data Q/A prep data and run ifsar\_review.py.

1. Obtain block imagery data from repository ifsar directory, [\\V480d.ftw.nrcs.usda.gov\qfs6](http://V480d.ftw.nrcs.usda.gov/qfs6), for review processing.
2. Mosaic ori, dsm, and dtm into 16blk quads into respective raster datasets. For dtm, eliminate the -10000 value; otherwise the derivative products will show those values.
3. Run Spatial Analyst surface tools to create derivative products of slope, hillshade and contours. Use conversion of 3.2808399 to convert meters to feet for display, and zdelta of > or < 1500 to determine contour interval.

For prepping and loading the data onto the Data Gateway:

1. Revise Product Description file using any new information at: <http://www.intermap.com/uploads/1170106364.pdf>
2. Run CatalogFP\_Maker for each product and generate the catalog shape files.

3. When there are new quads that are not already in the qd24kel index map such as quads along the Mexican border, these quads must be added to qd24kel. CatalogFP\_Maker hacks up a furball and manufactures these quads.
  - Select the quads based on the CatDesc tag !@mapindex from the fpDest map and save these features to a new map.
  - Load the .dbf from the new map into MS access. Open the imported table in design view and delete the Bytesize and DataPath fields.
  - Rename the CatID field QuadID. Rename the CatDesc field QuadName. Edit the QuadName field to be 'no\_USGS\_name' Edit the QuadID field to remove the trailing 'dsm' 'dtm' etc., remove the preceding 'n' and the 'w' in the middle. Set the field size for QuadID to 7.
  - Edit the county FIPS code(s) into the FIP\_C field.
  - Merge the new map with qd24kel by using ArcMap/toolbox/Data Management Tools/General/Merge. Merge the qd24kel map with the new map.
  - After examining, rename the merged map qd24kel, put it in fpSource and send the map to Fort Collins for storage and archiving.
  - Load the qd24kel.dbf in the table qd24kel in the zoneMBRdb.mdb MS access data base on all data service machines. Ensure that all the new quad rows are in the table.
4. Create the Status Maps for each product (link from "Status Maps" page) for each product.
5. Run MakePreviews for each product to generate the preview images and metadata for use in Step two of the gateway ordering process.
6. Notify gateway Fort Collins team to load the catalogs, status maps and news.

### 3. Technical Issues

- a. Tiling
- b. Compression  
GeoTiff
- c. Scale  
1:24,000
- d. Tonal Matching  
None
- e. Edge-matching  
None

### 4. Quality Control

- a. Procedures  
Visual quality checks using derived hillshade, slope, contours. The derived products are compared to other available data, including but not limited to high resolution imagery, DRG's, and other elevation products.

- b. Acceptance Criteria  
Visual quality checks using derived hillshade, slope, contours. Report findings within 60 days of receipt of data from Contractor; accept or reject within the 60-day timeframe.

### 5. Data Steward

- a. Name and Organization

Currently, the data steward for the received data is:  
National Cartography and Geospatial Center  
Natural Resources Conservation Service  
US Department of Agriculture  
501 Felix Street, Building 23  
P. O. Box 6567  
Fort Worth, Texas 76115-0567 USA

b. Responsibilities

Storage and access of the data.

**B. Integrated Data Structure**

**1. Geospatial Data Format**

b. Format (raster, vector, etc.)

Raster

c. Format Name

GeoTiff

d. Data Extent

Same as source data

e. Horizontal and Vertical Resolution

Same as source data

f. Absolute Horizontal and Vertical Accuracy

Same as source data

g. Nominal Scale

1:12,000

h. Horizontal and Vertical Datum

Same as source data

i. Projection

UTM

j. Coordinate Units

Meters

k. Symbology

None

**2. Attribute Data Format**

b. Format Name

N/A Raster data

- c. Database Size

Currently over 150GB, and will continue to grow as more data becomes available and is posted on the Data Gateway.

### 3. Data Model

- a. Geospatial Data Structure

GeoTiff

- b. Attribute Data Structure

N/A raster data

- c. Database Table Definition

N/A raster data

- d. Data Relationship Definition

N/A raster data

- e. Data Dictionary

N/A raster data

## **C. Resource Requirements**

- 1. Hardware and Software

This is unknown at this time.

- 2. Staffing

This is unknown at this time.

## **D. Integration Cost**

- 1. Hardware and Software

This is unknown at this time.

- 2. Staffing

This is unknown at this time.

## **IV. Delivery**

### **A. Specifications**

- 1. Directory Structure

- a. Folder Theme Data is Stored In

F:\geodata\elevation

- 2. File Naming Convention

<http://www.itc.nrcs.usda.gov/scdm/docs/SPG-GeospatialDataSetFileNamingStandard.pdf>

a. List of Theme Files and The File Naming Convention

COR	n<xx>w<yyy><a#>cor.ext
DSM	n<xx>w<yyy><a#>dsm.ext
DTM	n<xx>w<yyy><a#>dtm.ext
ORI	n<xx>w<yyy><a#>ori.ext

Where xx = latitude value; yyy = longitude value; a# = USGS MRC; .ext = the file extension.  
Example: n43w108c1cor.tif

**B. User Information**

1. Accuracy Assessment

a. Alignment with Other Theme Geospatial Data

This elevation data should be considered sufficiently detailed for the purpose of analysis at large scales within the limitations specified in the product handbook at <http://www.intermap.com/uploads/1170106364.pdf>. Alignment with the other data layers will not be perfect due to the fact that the data is captured at different scales and at different dates from other data.

b. Content

This elevation data should be considered sufficiently detailed for the purpose of analysis at large scales within the limitations specified in the product handbook at <http://www.intermap.com/uploads/1170106364.pdf>

2. Appropriate Uses of the Geospatial Data

a. Display Scale

For a hillshade image, use a scale of 1:12,000 or smaller.  
For a contour map, use a scale of 1:12,000 or smaller.

b. Plot Scale

For a hillshade image, the scale of 1:12,000 or smaller.  
For a contour map, the scale of 1:12,000 or smaller.

c. Area Calculations

Area Calculations are as accurate as the source data and capture scale and the algorithm used by ESRI software.

d. Decision Making

The data is as accurate as the source data and capture scale and the algorithm used by ESRI software

**C. Maintenance and Updating**

1. Recommendations and Guidelines

a. Original data location and structure

The integrated database is at NCGC and the data is delivered to the Service Center.

b. Update Cycle

As more IFSAR data is purchased from Intermap, it will become available on the Data Gateway.

c. Availability

As more IFSAR data is purchased from Intermap, it will become available on the Data Gateway.

d. Change Control

This is to be determined.

```
copyIFSAR.py
# -----
# this program will list all the raster datasets under a directory, then
# Calculate the UTMZone Number based on Lat/Long, then move/copy the related
# IFSAR rasters to respective directorires
#
# Input IFSAR data: \\v480d\qfs7\IFSAR\NMUSA_Block_4Digits (here the block is the
NextMap block for the whorld
#
# output Data: \\v480d\qfs1\ifsar\ifsarZone13(2digits)\ifsar32102(5digits latLong
block)\COR...ORI...DSM...DTM
#.....
#
# input test directory:C:\junk\copyIFSAR\NMUSA_Block_3201
# output Test directory: C:\junk\copyIFSAR\output Afterwards, create ifsar32102 and COR
ect. for the first time
#
#
# Original file format from InterMap*****
#****COR: Geotif and aux
#****ORI: Geotif, aux, html, txt, xml
#****DSM: ESRIGrid, html, txt, xml
#****DTM: ESRIGrid, html, txt, xml
#*****
# -----

# Import system modules

# Import system modules
import arcgisscripting
import sys, string, os
import math
import time
import os.path
import shutil

gp = arcgisscripting.create()

gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management
Tools.tbx")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Analysis Tools.tbx")
gp.CheckOutExtension("Spatial")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Spatial Analyst
Tools.tbx")

print 'start time for the copy process:', time.asctime()

all_start = time.time()

# Create the Geoprocessor object

# Load required toolboxes...
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Conversion Tools.tbx")

def utmZone(mrc_code,offset = 0.5):
    'Returns lat,lon,utmzone strings from mrc_code. Offset if using mrc code, moves to
left side.'
```

```
##print mrc_code
lat = '%s' % mrc_code[0:2]
##print lat
lon = '-%s' % mrc_code[2:5]
left_lon = (float(lon) - offset)
##
print lon
utmzone = '%i' % int(abs(abs(left_lon / 6) - 31))
##
print 'utmzone: ',utmzone
if len(utmzone) == 1:
    utmzone = '0' + utmzone
return lon,lat,utmzone

#IFSAR data location and create workspace for that location
##rasterWS = 'C:/junk/copyIFSAR/NMUSA_Block_3201'
##rasterWS = 'H:/NMUSA_Block_3187'
rasterWS = 'i:/blk3092'
#rasterWS = 'Q:/NMUSA_Block_3052_24Jan07_dv'
print 'i am here'
ws = rasterWS + '/COR'
gp.workspace = ws

#output directory
#outDir = 'C:/junk/copyIFSAR/output'
outDir = 'i:/IFSAR'
print 'i am here'

searchPattern = '*.tif'

#Going to list all the COR rasters
try:
    rasterLists = gp.listrasters(searchPattern)
except:
    print 'list Raster error', gp.GetMessages()

rasterLists.Reset()
rasterList = rasterLists.Next()

#Going to count the total quads for processing
totalQuads = 0
while rasterList:
    totalQuads = totalQuads + 1
    rasterList = rasterLists.Next()

print 'Total Quads need to be processed is ', totalQuads, 'for ', rasterWS
print

rasterLists.Reset()
rasterList = rasterLists.Next()

processedQuad = 0
#going to one on one check the raster and then copy to intended places
while rasterList:
    processedQuad = processedQuad + 1
    corRaster = ws + '/' + rasterList
##
print corRaster

#get raster quad Name
rasterQuadName = rasterList[0:9]
```

```
#going to get longitude and then calculate the utmzone
longitude = corRaster[-12:-9]
## print 'longitude:', longitude
#longFloat = float(longitude)
latitude = corRaster[-15:-13]
## print 'latitude:', latitude
#latFloat = float(latitude)
mrc_code = latitude + longitude
## print 'mrc_code is: ', mrc_code
lon,lat,utmzone = utmZone(mrc_code)
print 'current Quad is', rasterQuadName,' at utmz ',utmzone, ' ',processedQuad, 'out
of total ', totalQuads
#Check to see if the one degree block is exist
oneDirUp = outDir + '/' + 'ifsarZone' + utmzone
oneDirName = 'ifsar' + mrc_code
oneDDirectory = oneDirUp + '/' + oneDirName
## print 'oneDDirectory:', oneDDirectory

if not gp.exists(oneDDirectory):
## print 'going to create One degree block directory'
try:
    gp.CreateFolder_management(oneDirUp, oneDirName)
    gp.refreshcatalog(oneDirUp)
except:
    print 'Create one degree block directory error:', gp.GetMessages()

#going to create the 4 type directory
try:
    gp.CreateFolder_management(oneDDirectory, "COR")
    gp.CreateFolder_management(oneDDirectory, "DSM")
    gp.CreateFolder_management(oneDDirectory, "DTM")
    gp.CreateFolder_management(oneDDirectory, "ORI")
    gp.refreshcatalog(oneDDirectory)
except:
    print 'Create the 4 Type subDirectory error:', gp.GetMessages()

#going to copy/move each individual IFSAR data to the respective position
inCOR = rasterWS + '/COR/' + rasterQuadName + 'cor.tif'
inORI = rasterWS + '/ORI/' + rasterQuadName + 'ori.tif'
## inCORAux = rasterWS + '/COR/' + rasterQuadName + 'cor.aux'
## inORIAux = rasterWS + '/ORI/' + rasterQuadName + 'ori.aux'

inDSM = rasterWS + '/DSM/' + rasterQuadName + 'dsm'
inDTM = rasterWS + '/DTM/' + rasterQuadName + 'dtm'

inORITxt = rasterWS + '/ORI/' + rasterQuadName + 'ori.txt'
inDSMTxt = rasterWS + '/DSM/' + rasterQuadName + 'dsm.txt'
inDTMTxt = rasterWS + '/DTM/' + rasterQuadName + 'dtm.txt'

inORIHtml = rasterWS + '/ORI/' + rasterQuadName + 'ori.html'
inDSMHtml = rasterWS + '/DSM/' + rasterQuadName + 'dsm.html'
inDTMHtml = rasterWS + '/DTM/' + rasterQuadName + 'dtm.html'

inORIXml = rasterWS + '/ORI/' + rasterQuadName + 'ori.xml'
inDSMXml = rasterWS + '/DSM/' + rasterQuadName + 'dsm.xml'
inDTMXml = rasterWS + '/DTM/' + rasterQuadName + 'dtm.xml'
```

```
rasterQuadNameLow = rasterQuadName.lower()

outCOR = oneDDirectory + '/COR/' + rasterQuadNameLow + 'cor.tif'
outORI = oneDDirectory + '/ORI/' + rasterQuadNameLow + 'ori.tif'
outDSM = oneDDirectory + '/DSM/' + rasterQuadNameLow + 'dsm.tif'
outDTM = oneDDirectory + '/DTM/' + rasterQuadNameLow + 'dtm.tif'

outCORAux = oneDDirectory + '/COR/' + rasterQuadNameLow + 'cor.aux'
outORIAux = oneDDirectory + '/ORI/' + rasterQuadNameLow + 'ori.aux'

outORITxt = oneDDirectory + '/ORI/' + rasterQuadNameLow + 'ori.txt'
outDSMTxt = oneDDirectory + '/DSM/' + rasterQuadNameLow + 'dsm.txt'
outDTMTxt = oneDDirectory + '/DTM/' + rasterQuadNameLow + 'dtm.txt'

outORHtml = oneDDirectory + '/ORI/' + rasterQuadNameLow + 'ori.html'
outDSMHtml = oneDDirectory + '/DSM/' + rasterQuadNameLow + 'dsm.html'
outDTMHtml = oneDDirectory + '/DTM/' + rasterQuadNameLow + 'dtm.html'

outORIXml = oneDDirectory + '/ORI/' + rasterQuadNameLow + 'ori.xml'
outDSMXml = oneDDirectory + '/DSM/' + rasterQuadNameLow + 'dsm.xml'
outDTMXml = oneDDirectory + '/DTM/' + rasterQuadNameLow + 'dtm.xml'

#use copyfile to directly copy geoTif and aux files for COR & ORI
#MODIFICATION: shutil copyfile does NOT create .tif.xml for COR or ORI.
# Therefore, script has been modified to use CopyRaster_management.

# shutil.copyfile(inCOR, outCOR)
# shutil.copyfile(inORI, outORI)
gp.overwriteoutput = 1

gp.CopyRaster_management(inCOR, outCOR, "", "", "", "", "", "")
gp.CopyRaster_management(inORI, outORI, "", "", "", "", "", "")
## shutil.copyfile(inCORAux, outCORAux)
## shutil.copyfile(inORIAux, outORIAux)

#use setNon to set -10000 elevation to ESRI NoData for DSM and DTM

#gp.SetNull_sa(inDSM, inDSM, outDSM, "value = -10000")
#gp.SetNull_sa(inDTM, inDTM, outDTM, "value = -10000")

gp.CopyRaster_management(inDSM, outDSM, "", "", "", "", "", "")
gp.CopyRaster_management(inDTM, outDTM, "", "", "", "", "", "")

shutil.copyfile(inORITxt, outORITxt)
shutil.copyfile(inDSMTxt, outDSMTxt)
shutil.copyfile(inDTMTxt, outDTMTxt)

shutil.copyfile(inORHtml, outORHtml)
shutil.copyfile(inDSMHtml, outDSMHtml)
shutil.copyfile(inDTMHtml, outDTMHtml)

shutil.copyfile(inORIXml, outORIXml)
shutil.copyfile(inDSMXml, outDSMXml)
shutil.copyfile(inDTMXml, outDTMXml)

gp.refreshcatalog(oneDDirectory)
```

```
#after refresh, calculate stats and build pyramids for the COR and ORI. Basically to
create the .aux an rrd file
gp.CalculateStatistics_management (outCOR, 1, 1, "")
gp.BuildPyramids_management (outCOR)

gp.CalculateStatistics_management (outORI, 1, 1, "")
gp.BuildPyramids_management (outORI)
gp.refreshcatalog(oneDDirectory)

#print 'print to stop the program', printStop

rasterList = rasterLists.Next()

print 'total time used is: ', (time.time()- all_start)/60, ' Minutes for processing ', rasterWS

sys.exit()
```

ifsar\_review.py

'''

\*\*\*\*\*

This program will take 7.5min Quad IFSAR data and create an ori mosaic, a dsm mosaic, and a dtm mosaic from which derived products contours, slopes, and hillshades, are produced, for quality control and acceptance of the ifsar data.

Required directory structure and data to run program:

<drive>:/ifsar/ifsar.shp with required fields which need to be added to attribute table:

STATUS Short 2  
DSMPATH Text 82  
DTMPATH Text 82  
ORIPATH Text 82  
MRC\_MOSAIC Text 7

<drive>:/ifsar/idx\_mosaic.shp

<drive:::/ifsar/idx\_ned.shp

<drive>:/blk<your copied block from qfs6>

Generated products of 3 mosaics in geodatabases, with derived products in dtm geodatabase. See this Example:

<drive>:/ifsar/n47098  
dsm47098a5  
d47098a5  
dtm47098a5  
d4798a5  
sp47098a5  
con\_47098a5  
hs47098a5  
ori47098a5  
o47098a5

\*\*\*\*\*

Modification Record:

October 16, 2006. decided to create hill shade directly from 1 degree block of NED10 or resampled NED30. The database used is idx\_hillshade. Run def hillshade to create 1 degree block of hillshade.

\*\*\*\*\*

'''

```
import arcgisscripting
import sys, string, os
import time
import os.path
```

```
gp = arcgisscripting.create()
# Create the Geoprocessor object
all_start = time.time()
print 'collection statistics time for all files: ', time.asctime()
#gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")
```

```
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Analysis Tools.tbx")
gp.CheckOutExtension("Spatial")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Spatial Analyst
Tools.tbx")
gp.scratchWorkspace = 'i:\\temp'
gp.workspace = "i:\\ifsar"

global root_dir
root_dir = gp.workspace

def findStats(utmzone,mrc_code):

    blk_dir = root_dir + "\\ " + "n" + mrc_code[0:5]
    RasterDS = "d" + mrc_code
    dtmwksp = "dtm" + mrc_code
    ext = ".mdb"
    BlkLoc = blk_dir + "\\ " + dtmwksp + ext
    TargetRaster = blk_dir + "\\ " + dtmwksp + ext + "\\ " + RasterDS

    OutStats = root_dir + "\\ " + "st" + mrc_code + ".txt"
    if gp.exists(TargetRaster):

        try:
            gp.BandCollectionStats_sa(TargetRaster, OutStats, "BRIEF")
            print TargetRaster + ' successful'
        except:
            print 'error Stats ' + TargetRaster
        try:
            getStats(utmzone,mrc_code,OutStats)
        except:
            print 'error'

def getStats(utmzone, mrc_code, OutStats):
    print 'getStats'

    mainfile = root_dir + "\\ " + "ifsar.txt"
    print mainfile
    if gp.exists(mainfile):
        outfile = open(mainfile, "a")
    else:
        outfile = open(mainfile, "w")

    count = 1
    infile = open (OutStats, "r")
    for line in infile.readlines ():

        print "Line",count,line,
        line = string.strip (line)
        print line
        theData = string.split (line, "\\t")
        print count, theData
        test = line.split()
        print test
        if count == 7:

            test = line.split()
```

```
del test[0]
mrc = mrc_code.replace(""," ")
print 'this is mrcs ' + mrc
test.append(mrc)
rec = string.join(test)
print rec
min = float(test[0])
print str(min)
max = float(test[1])
print str(max)
zdelta = max - min
print 'this is zdelta ' + str(zdelta)
outfile.write(rec + "\n")

count = count + 1
infile.close()
outfile.close()
print 'this is delta ' + str(zdelta)
createContour(utmzone, mrc_code, zdelta)

def createContour(utmzone, mrc_code, zdelta):
    print 'countour time ...'
    # Local variables...
    start = time.time()
    cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)
    #gp.workspace = "i:\\ifsar"
    blk = "n" + mrc_code[0:5]
    mwrk = "dtm" + mrc_code + ".mdb"
    RasterWS = root_dir + "\\\" + blk + "\\\" + mwrk
    gp.refreshcatalog(RasterWS)

    InputRaster = RasterWS + "\\\" + "d" + mrc_code

    #OutContour = "con" + mrc_code
    OutContour = RasterWS + "\\\" + "con_" + mrc_code

    Zfactor = 3.2808399
    #print str(Zfactor)
    meters_to_feet = 0.304800609601
    base_offset = 0.00000000001
    BaseContour = 0
    #if int(elev_delta) < 1500:

    if zdelta < 1500:

        #contour_interval = "%1.10f" % (10 * meters_to_feet)
        ContourInterval = 20

    else:

        ContourInterval = 50

    print "the next step may take some time"

    # Process: Contour
```

```
if not gp.exists(OutContour):
    try:
        #print "trying to Create OutContour " + OutContour
        gp.RasCon(Zfactor, InputRaster, ContourInterval, cs, BaseContour, OutContour)
        print "Successful " + OutContour

    except:
        #print error message if an error occurs
        print gp.AddMessage(gp.GetMessages(2))

try:
    gp.refreshcatalog(RasterWS)

except:
    print gp.AddMessage(gp.GetMessages(2))
    print 'Contour time: ', (time.time() -start) / 2

def createHillshade(utmzone, mrc_code):

    cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)

    blk = "n" + mrc_code[0:5]
    mwrk = "dtm" + mrc_code + ".mdb"
    RasterWS = root_dir + "\\ " + blk + "\\ " + mwrk
    gp.refreshcatalog(RasterWS)

    InputRaster = RasterWS + "\\ " + "d" + mrc_code
    #print InputRaster

    HS_Mosaic = RasterWS + "\\ " + "hs" + mrc_code
    #print "this is HS_Mosaic " + HS_Mosaic

    print "working"
    start = time.time()

    print "this will take some time....about 15 minutes...to filter and create hillshade"

    #Process Hillshade
    if not gp.exists(HS_Mosaic):
        try:

            #gp.Hillshade_sa(InputRaster, OutputRaster, "315", "45", "NO_SHADOWS", "1")
            gp.RasHS(InputRaster, HS_Mosaic)
            #According to ESRI article 29366, it will be better to use 0.000011 as the Z value
            print "successful hillshade " + HS_Mosaic
        except:

            print gp.AddMessage(gp.GetMessages(2))

    gp.RefreshCatalog(RasterWS)

def createSlope(utmzone, mrc_code):
```

```
#ma("slopeg = slope(elev, 0.3048)")

print "working"
cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)

start = time.time()
#print (start)
gp.AddMessage(start)

#Local Variables

blk = "n" + mrc_code[0:5]
mwrk = "dtm" + mrc_code + ".mdb"
RasterWS = root_dir + "\\ " + blk + "\\ " + mwrk
gp.refreshcatalog(RasterWS)

InputRaster = RasterWS + "\\ " + "d" + mrc_code
#InRaster = InputRaster

OutRaster = RasterWS + "\\ " + "sp" + mrc_code

gp.refreshcatalog(RasterWS)

#print OutRaster
OutMeasurement= "PERCENT_RISE"
#print InMeasurementType
Zfactor = 1
tempEnvironment0 = cs
cs = tempEnvironment0
# Process: Slope

if not gp.exists(OutRaster):
    try:

        gp.Slope_sa(InputRaster, OutRaster, "PERCENT_RISE", Zfactor)
        #cs = tempEnvironment0
        #gp.Slope_sa(InputRaster, OutputRaster, OutputMeasurement, Zfactor)

        #gp.RasSL(InputRaster, Zfactor, OutMeasurement, cs, OutputRaster)
        print "successful " + OutRaster
    except:
        print "error"
gp.refreshcatalog(RasterWS)
gp.AddMessage('Slope time for creating this slope in Minutes:')
print gp.AddMessage((time.time() - start) / 60)

def makeOri(OutFC, utmzone, mrc_code):

    gp.overwriteoutput = 1

    cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)
```

```
print "MOSAICKING ORTO TIFS WILL TAKE AT LEAST 20 -30 Minutes FOR 16  
QUADS... BE PATIENT!~!!!!!!"
```

```
all_start = time.time()  
print 'contour generation start time: ', time.asctime()  
blk = "n" + mrc_code[0:5]  
mwrk = "ori" + mrc_code + ".mdb"  
RasterWS = root_dir + "\\ " + blk + "\\ " + mwrk
```

```
gp.refreshcatalog(RasterWS)
```

```
#print "This is the outshape " + OutFC  
orifld = "ORIPATH"  
#print orifld
```

```
TargetRaster = RasterWS + "\\ " + "o" + mrc_code  
print TargetRaster  
#TargetRaster = r"i:\ifsar\n33100a5\ori.mdb\u33100a5"
```

```
vtab = gp.CreateObject("ValueTable", 1)  
print "\n" + "Querying "  
#gp.AddMessage("\n" + "Querying " + InFC + " using " + Expression)
```

```
try:  
    rows = gp.SearchCursor(OutFC)  
    row = rows.Next()  
    while row:  
        aVal = row.GetValue(orifld)  
        print aVal  
        vtab.AddRow(aVal)  
        row = rows.Next()
```

```
except:  
    print gp.GetMessages(2)
```

```
print vtab.Exporttostring()
```

```
try:  
  
    gp.RefreshCatalog(RasterWS)  
    print "successful" + RasterWS  
except:  
    print "error"
```

```
print "BE PATIENT.....NOW COMES THE REAL LONG PART...."
```

```
try:  
  
    gp.Mosaic_management(vtab, TargetRaster, "LAST","FIRST", "", "", "NONE", "0")  
    print "mosaicking is successful" + TargetRaster  
except:  
    print gp.GetMessages(2)
```

```
gp.refreshcatalog(RasterWS)
```

```
def makeDSM(OutFC, utmzone, mrc_code):  
    print "starting makedsm function...."
```

```
print OutFC
blk_dir = root_dir + "\\\" + \"n\" + mrc_code[0:5]
RasterDS = \"d\" + mrc_code
dsmwksp = \"dsm\" + mrc_code
ext = \".mdb\"
BlkLoc = blk_dir + "\\\" + dsmwksp + ext
print BlkLoc

cs =
\"GEOGCS['GCS_North_American_1983',DATUM['D_North_American_1983',SPHEROID['
GRS_1980',6378137.0,298.257222101]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.017453
2925199433]]\"
print \"MOSAICKING DTMS WILL TAKE ABOUT 8 minutes FOR 16 QUADS... BE
PATIENT!~!!!!!!\"

all_start = time.time()
print 'mosaic generation start time: ', time.asctime()
dsmfld = \"DSMPATH\"
print dsmfld

TargetRaster = blk_dir + "\\\" + dsmwksp + ext + "\\\" + RasterDS
print \"This is TargetRaster \" + TargetRaster

vtab = gp.CreateObject(\"ValueTable\", 1)
print \"\n\" + \"Querying \"
gp.AddMessage(\"\\n\" + \"Querying \" + OutFC)

try:
    rows = gp.SearchCursor(OutFC)
    row = rows.Next()
    while row:
        aVal = row.GetValue(dsmfld)
        print aVal
        vtab.AddRow(aVal)
        row = rows.Next()

except:
    print gp.GetMessages(2)

print vtab.Exporttostring()

try:
    print \"trying to refresh catalog \" + blk_dir
    #gp.RefreshCatalog(gp.workspace)
    gp.RefreshCatalog(blk_dir)
    print \"successful\"
except:
    print \"error\"

print \"BE PATIENT.....NOW COMES THE REAL LONG PART....\"
try:
    print \"trying to mosaic\"
    #gp.Mosaic_management(vtab, TargetRaster, \"LAST\", \"FIRST\", \"-10000\", \"0\",
\"NONE\", \"\")
    #gp.Mosaic_management(vtab, TargetRaster, \"LAST\", \"FIRST\", \"\", \"0\", \"NONE\", \"\")
    gp.Mosaic_management(vtab, TargetRaster, \"LAST\", \"FIRST\", \"\", \"\", \"NONE\", \"0\")
    print \"mosaicking is successful\"
```

```
except:
    print gp.GetMessages(2)

def makeDtm(OutFC, utmzone, mrc_code):

    #local variables
    print 'dtm ' + utmzone
    gp.overwriteoutput = 1
    print "starting makedtm function...."
    blk_dir = root_dir + "\\\" + "n" + mrc_code[0:5]
    RasterDS = "d" + mrc_code
    dtmwksp = "dtm" + mrc_code
    ext = ".mdb"
    BlkLoc = blk_dir + "\\\" + dtmwksp + ext

    print "MOSAIKING DTMS WILL TAKE ABOUT 8 minutes FOR 16 QUADS... BE
PATIENT!~!!!!!"

    all_start = time.time()
    print 'mosaic generation start time: ', time.asctime()
    dtmfld = "DTMPATH"
    print dtmfld

    TargetRaster = blk_dir + "\\\" + dtmwksp + ext + "\\\" + RasterDS
    print "This is TargetRaster " + TargetRaster

    vtab = gp.CreateObject("ValueTable", 1)
    print "\n" + "Querying "
    gp.AddMessage("\n" + "Querying " + OutFC)

    try:
        rows = gp.SearchCursor(OutFC)
        row = rows.Next()
        while row:
            aVal = row.GetValue(dtmfld)
            print aVal
            vtab.AddRow(aVal)
            row = rows.Next()

    except:
        print gp.GetMessages(2)

    print vtab.Exporttostring()

    try:
        print "trying to refresh catalog " + blk_dir
        #gp.RefreshCatalog(gp.workspace)
        gp.RefreshCatalog(blk_dir)
        print "successful"
    except:
        print "error"

    print "BE PATIENT.....NOW COMES THE REAL LONG PART...."
    InPropertyType = "ROWCOUNT"
    rc = gp.GetRasterProperties (TargetRaster, InPropertyType)
```

```
try:
    print "trying to mosaic"

    gp.Mosaic_management(vtab, TargetRaster, "LAST","FIRST", "-10000", "0", "NONE",
    "")
    print "mosaicking is successful"
except:
    print gp.GetMessages(2)

#def verifyPath(utmzone, mrc_code):
def verifyPath(mrc_code):
    #gp.workspace = "i:\ifsar"
    root_dir = gp.workspace

    #gp.overwriteoutput = 1
    blkbox = mrc_code[0:5]
    blk_dir = root_dir + "\\\" + "n" + blkbox
    ext = ".mdb"
    dtmwksp = blk_dir + "\\dtm" + mrc_code + ext
    oriwksp = blk_dir + "\\ori" + mrc_code + ext
    print 'this is oriwksp ' + oriwksp

    #cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate
Systems/Utm/Nad 1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)

    if not gp.exists(blk_dir):
        try:
            print "trying to create " + blk_dir
            gp.CreateFolder(gp.workspace, "n" + blkbox)
            print "successful"
        except:
            print "error"
    if not gp.exists(oriwksp):

        try:

            gp.CreatePersonalGDB(os.path.dirname(oriwksp),os.path.basename(oriwksp))
            print "this is successful" + os.path.dirname(oriwksp) + "" + os.path.basename(oriwksp)

        except:

            print "error"
            print gp.AddMessage(gp.GetMessages(2))
        else:

            print "exists " + oriwksp

    gp.refreshcatalog(blk_dir)

    if not gp.exists(dtmwksp):

        try:

            gp.CreatePersonalGDB(os.path.dirname(dtmwksp),os.path.basename(dtmwksp))
            print "this is successful"
```

```
except:
    print gp.AddMessage(gp.GetMessages(2))
else:

    print "exists " + dtmwksp

gp.refreshcatalog(blk_dir)
print 'i am leaving verifypath '

#sel_mrc(root_dir, utmzone, blkfld)

def crDTMDS(utmzone, mrc_code):

    gp.overwriteoutput = 1
    #local variables
    print "crDTMDS function"
    blk_dir = root_dir + "\\ " + "n" + mrc_code[0:5]
    RasterDS = "d" + mrc_code
    dtmwksp = "dtm" + mrc_code
    ext = ".mdb"
    BlkLoc = blk_dir + "\\ " + dtmwksp + ext
    RasDS = BlkLoc + "\\ " + RasterDS
    cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)
    Pixel_Type = "32_BIT_FLOAT"

    all_start = time.time()

    if not gp.exists(RasDS):
        try:

            gp.CreateRasterDataset_management(BlkLoc, RasterDS, "", Pixel_Type, cs, "1", "",
"#")
            print "this is successful " + RasterDS
        except:
            print gp.AddMessage(gp.GetMessages(2))

def crORIDS(utmzone, mrc_code):

    gp.overwriteoutput = 1

    #local variables
    print "crORIDS function"
    blk_dir = root_dir + "\\ " + "n" + mrc_code[0:5]
    RasterDS = "o" + mrc_code
    oriwksp = "ori" + mrc_code
    ext = ".mdb"

    BlkLoc = blk_dir + "\\ " + oriwksp + ext
    RasDS = BlkLoc + "\\ " + RasterDS
```

```
cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)
Pixel_Type = "8_BIT_UNSIGNED"
```

```
if not gp.exists(RasDS):
```

```
    try:
```

```
        gp.CreateRasterDataset_management(BlkLoc, RasterDS, "", Pixel_Type, cs, "1", "",
"#")
        print "this is successful " + RasterDS
```

```
    except:
```

```
        print gp.AddMessage(gp.GetMessages(2))
```

```
def crDSMS(utmzone, mrc_code):
```

```
    gp.overwriteoutput = 1
    #local variables
    print "crDSMS function"
    blk_dir = root_dir + "\\ " + "n" + mrc_code[0:5]
    print 'this is blk_dir ' + blk_dir
    RasterDS = "d" + mrc_code
    dsmwksp = "dsm" + mrc_code
    ext = ".mdb"
    dsmwkrsp = dsmwksp + ext
    print 'this is dsmwkrsp ' + dsmwkrsp
    BlkLoc = blk_dir + "\\ " + dsmwkrsp
    print 'this is blkLoc ' + BlkLoc
    RasDS = BlkLoc + "\\ " + RasterDS
    print 'this is RasDS ' + RasDS
```

```
cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)
Pixel_Type = "32_BIT_FLOAT"
```

```
all_start = time.time()
```

```
if not gp.exists(BlkLoc):
```

```
    try:
```

```
        gp.CreatePersonalGDB(blk_dir, dsmwkrsp)
```

```
    except:
```

```
        print gp.AddMessage(gp.GetMessages(2))
```

```
if not gp.exists(RasDS):
```

```
    try:
```

```
        print 'trying to create dsm raster'
```

```
        gp.CreateRasterDataset_management(BlkLoc, RasterDS, "", Pixel_Type, cs, "1", "",
"#")
```

```
        print "this is successful " + RasterDS
```

```
    except:
```

```
        print gp.AddMessage(gp.GetMessages(2))
```

```
def cr_main(utmzone,mrc_code):
```

```
print 'this is cr_main '

gp.overwriteoutput = 1

Mytoolbox = r"c:\ifsar\ifsartools\ifsar.tbx"
print Mytoolbox

#Mytbx = r"i:\ifsar\ifsartools\mytbx.tbx"
gp.AddToolbox(Mytoolbox)
#gp.AddToolbox(Mytbx)
#Check usage
print gp.Usage("RASblk")
print 'i am at cr_main '
#print gp.Usage("RasHS")
#print gp.Usage("RasCon")
#print gp.Usage("RasSL")

#LOCAL VARIABLES

blkexp = "\"MRC_MOSAIC\" = '%s'" % (mrc_code)
#blkexp = "\"MRC_MOSAIC\" = '%s' AND \"STATUS\" = 1"
print blkexp

#BlkFC = "Database
Connections\ncgcmartbase@sqlsde1.sde\ncgcmart.BASE.idx_30x30m"
#BlkFC = root_dir + "\" + "idx30x30m.shp"
InFC = root_dir + "\" + "ifsar.shp"
OutGeo = root_dir + "\" + "g" + mrc_code + ".shp"
print 'this is OutGeo ' + OutGeo
blk = "n" + mrc_code[0:5]

OutFC = root_dir + "\" + "n" + mrc_code + ".shp"
print 'this is OutFC ' + OutFC
#cs =
"GEOGCS['GCS_North_American_1983',DATUM['D_North_American_1983',SPHEROID['
GRS_1980',6378137.0,298.257222101]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.017453
2925199433]]"
#cs = "C:\Program Files\ArcGIS\Coordinate Systems\Geographic Coordinate
Systems\North America\North American Datum 1983.prj"

cs = "c:/Program Files/ArcGIS/Coordinate Systems/Projected Coordinate Systems/Utm/Nad
1983/NAD 1983 UTM Zone %sN.prj" % (utmzone)
print cs

if not gp.exists(OutFC):

    try:
        print "running model..."
        gp.RASblk(InFC, blkexp, OutGeo, cs, OutFC)
        print "creating " + OutFC + " successful"
    except:
        print gp.AddMessage(2)
        print "CReating " + OutFC + " not successful"

else:
```

```
print "The feature class exists " + OutFC

crDTMDS(utmzone, mrc_code)
#crORIDS(utmzone, mrc_code)
#crDSMS(utmzone, mrc_code)
#makeOri(OutFC, utmzone, mrc_code)
makeDtm(OutFC, utmzone, mrc_code)
#makeDSM(OutFC, utmzone, mrc_code)
createHillshade(utmzone, mrc_code)
createSlope(utmzone, mrc_code)
findStats(utmzone,mrc_code)
#createContour(utmzone, mrc_code)

def sel_mrc(root_dir, utmzone, mrc_code):

    print "here ifsar params"
    ncg_c_connect_str = IFSARParams.ncgc_connect_str
    print ncg_c_connect_str
    #Select * From Win32_Directory Where FileName LIKE 'Scripts-%'

    #sqlstr1 = "SELECT first,last,house,street,district,town,postcode,phone
    # FROM Address WHERE %s LIKE "%s"" % (field,value)

    #test = blkfld + "%"
    sql_str = "SELECT mrc_code FROM idx30x30 WHERE mrc_code LIKE '" + blkfld +
    "%'"
    #sql_str = "SELECT count(*) FROM idx30x30 WHERE mrc_code LIKE '" + blkfld +
    "%'"
    print sql_str
    #sql_str = "SELECT count(*) FROM ncg.ncgc.index30m WHERE mrc_code = '%s'" %
    blkfld

    #sql_str = "SELECT blk_name, utmzoneone,dbaflag FROM idx_hillshade WHERE
    utmzoneone = '%s' and dbaflag = 102 ORDER BY blk_name" % utmzone

    #sql_str1 = "SELECT count(*) FROM idx_hillshade WHERE utmzone utmzone= '%s' " %
    utmzone
    print sql_str
    blkList = Utilities.readSql(IFSARParams.ncgc_params,sql_str)
    count = 0
    for blk_rec in blkList:
        count = count + 1
        print blk_rec
        print str(count)

def main(status):

    gp.overwriteoutput = 1

    print 'i am here'

    Mytoolbox = r"c:\ifsar\ifsartools\ifsar.tbx"
    gp.AddToolbox(Mytoolbox)
    #Check usage
```

```
#print gp.Usage("RasIDX")
#print gp.Usage("RASblks")
#nm_exp = "\"UTMZONE\" = 13 AND \"STATUS\" = 1"

#nm_exp = "\"UTMZONE\" = %s AND \"STATUS\" = %s" % (utmzone,status)

nm_exp = "\"STATUS\" = %s" % (status)
print nm_exp
#print IDXExp

#QIDX48 = "Database Connections\\base@ncgcsde1a.sde\\base.BASE.QD1DEG48"
#BlkFC = "Database
Connections\\ncgcmartbase@sqlsde1.sde\\ncgcmart.BASE.idx_30x30m"

#BlkFC = root_dir + "\\\" + "idx30x30m.shp"
InFC = root_dir + "\\\" + "ifsar.shp"

OutFC = root_dir + "\\\" + "selall" + ".shp"
FreqFld = "MRC_MOSAIC"

blk_freq = gp.workspace + "\\\" + "frqnew" + ".dbf"
if gp.exists(OutFC):
    print 'exists ' + OutFC

if not gp.exists(blk_freq):

    try:
        print "trying to run model "
        gp.RasIDX(InFC, nm_exp, OutFC, blk_freq, FreqFld)
        print "successful"
    except:
        print "model not working"

if gp.exists(blk_freq):

    try:

        rows = gp.SearchCursor(blk_freq)
        row = rows.Next()

        while row:

            mrc_code = row.GetValue(FreqFld)
            print mrc_code
            verifyPath(mrc_code)
            lon,lat,utmzone = utmZone(mrc_code)
            print utmzone
            cr_main(utmzone,mrc_code)
            row = rows.Next()

    except:

        print gp.AddMessage(gp. GetMessages(2))
```

```
def cleanup():
    Mytoolbox = r"i:\ifsar\ifsartools\Ifsar.tbx"
    gp.removeToolbox(Mytoolbox)

def utmZone(mrc_code,offset = 0.5):
    'Returns lat,lon,utmzone strings from mrc_code. Offset if using mrc code, moves to
    left side.'
    #TODO: handle 04
    ##print mrc_code
    lat = '%s' % mrc_code[0:2]
    ##print lat
    lon = '-%s' % mrc_code[2:5]
    left_lon = (float(lon) - offset)
    ##print lon
    utmzone = '%i' % int(abs(abs(left_lon / 6) - 31))
    print 'utmzone: ',utmzone
    if len(utmzone) == 1:
        utmzone = '0' + utmzone
    return lon,lat,utmzone

#main("3056")
main(1)
```