
MISR Toolkit Python Interface

June 19, 2020

Abstract

This document describes the installation and usage of the Python interface to the MISR Toolkit.

1 Installation

Prerequisites:

1. [Python 2.7 or 3.x](#)
2. [MISR Toolkit](#)
3. [NumPy](#)

Building:

```
cd Mtk/wrappers/python
python setup.py install
```

2 Main Classes

2.1 MtkFile

class MtkFile (*filename*)

Constructs a new MtkFile object.

```
>>> m = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
```

attr_get (*attribute*)

Get a file attribute.

```
>>> m.attr_get('Path_number')
37
```

attr_list

List of file attributes names.

```
>>> m.attr_list
['HDFEOSVersion', 'StructMetadata.0', 'Path_number', 'AGP_version_id',
 'DID_version_id', 'Number_blocks', 'Ocean_blocks_size', 'Ocean_blocks.count',
 'Ocean_blocks.numbers', 'SOM_parameters.som_ellipsoid.a',
 'SOM_parameters.som_ellipsoid.e2', 'SOM_parameters.som_orbit.aprime',
 'SOM_parameters.som_orbit.eprime', 'SOM_parameters.som_orbit.gama',
 'SOM_parameters.som_orbit.nrev', 'SOM_parameters.som_orbit.ro',
 'SOM_parameters.som_orbit.i', 'SOM_parameters.som_orbit.P2P1',
 'SOM_parameters.som_orbit.lambda0', 'Origin_block.ulc.x',
 'Origin_block.ulc.y', 'Origin_block.lrc.x', 'Origin_block.lrc.y',
 'Start_block', 'End_block', 'Cam_mode', 'Num_local_modes',
 'Local_mode_site_name', 'Orbit_QA', 'Camera', 'coremetadata']
```

block

Start and end block numbers.

```
>>> m.block
(1, 140)
```

block_metadata_list

List of block metadata structure names.

```
>>> m.block_metadata_list
['PerBlockMetadataCommon', 'PerBlockMetadataRad', 'PerBlockMetadataTime']
```

block_metadata_field_list (*blockmetaname*)

List fields in a block metadata structure.

```
>>> m.block_metadata_field_list('PerBlockMetadataCommon')
['Block_number', 'Ocean_flag', 'Block_coor_ulc_som_meter.x',
 'Block_coor_ulc_som_meter.y', 'Block_coor_lrc_som_meter.x',
 'Block_coor_lrc_som_meter.y', 'Data_flag']
```

block_metadata_field_read (*blockmetaname, fieldname*)

Read a block metadata field.

```
>>> m.block_metadata_field_read('PerBlockMetadataCommon', 'Block_number')
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140]
```

core_metadata_get (*parameter*)

Get core metadata parameter.

```
>>> m.core_metadata_get('LOCALGRANULEID')
'MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf'
```

core_metadata_list

List of core metadata parameter names.

```
>>> m.core_metadata_list
['LOCALGRANULEID', 'PRODUCTIONDATETIME', 'LOCALVERSIONID', 'PGEVERSION',
 'MEASUREDPARAMETERCONTAINER', 'AUTOMATICQUALITYFLAGEXPLANATION',
 'AUTOMATICQUALITYFLAG', 'QAPERCENTMISSINGDATA', 'PARAMETERNAME',
 'ORBITCALCULATEDSPATIALDOMAINCONTAINER', 'EQUATORCROSSINGDATE',
 'EQUATORCROSSINGTIME', 'ORBITNUMBER', 'EQUATORCROSSINGLONGITUDE',
 'VERSIONID', 'SHORTNAME', 'INPUTPOINTER', 'GPOLYGONCONTAINER',
 'GRINGPOINTLONGITUDE', 'GRINGPOINTLATITUDE', 'GRINGPOINTSEQUENCENO',
 'EXCLUSIONGRINGFLAG', 'RANGEENDINGDATE', 'RANGEENDINGTIME',
 'RANGEBEGINNINGDATE', 'RANGEBEGINNINGTIME', 'ADDITIONALATTRIBUTESCONTAINER',
 'ADDITIONALATTRIBUTENAME', 'PARAMETERVALUE', 'ADDITIONALATTRIBUTESCONTAINER',
 'ADDITIONALATTRIBUTENAME', 'PARAMETERVALUE', 'ADDITIONALATTRIBUTESCONTAINER',
 'ADDITIONALATTRIBUTENAME', 'PARAMETERVALUE', 'ADDITIONALATTRIBUTESCONTAINER',
 'ADDITIONALATTRIBUTENAME', 'PARAMETERVALUE',
 'ASSOCIATEDPLATFORMINSTRUMENTSENSORCONTAINER',
 'ASSOCIATEDSENSORSHORTNAME', 'ASSOCIATEDPLATFORMSHORTNAME', 'OPERATIONMODE',
 'ASSOCIATEDINSTRUMENTSHORTNAME']
```

file_name

File name of file.

```
>>> m.file_name
'../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf'
```

file_type

MISR product file type.

```
>>> m.file_type
'GRP_ELLIPSOID_GM'
```

grid (*grid_name*)

Returns MtkGrid object for *grid_name*.

```
>>> m.grid('BlueBand')
<MisrToolkit.MtkGrid object at 0x140e740>
```

grid_list

List of grid names.

```
>>> m.grid_list
['BlueBand', 'GreenBand', 'RedBand', 'NIRBand', 'BRF Conversion Factors', 'GeometricParameters']
```

local_granule_id

Local granule ID of MISR product file.

```
>>> m.local_granule_id
'MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf'
```

orbit

Orbit number.

```
>>> m.orbit
29058
```

path

Path number.

```
>>> m.path
37
```

time_metadata_read()

Read time metadata from L1B2 Ellipsoid product.

```
>>> m.time_metadata_read()
<MisrToolkit.MtkTimeMetaData object at 0x158f000>
```

Note: Time metadata is stored in the L1B2 Ellipsoid product, it is not available in any other product.

version

MISR product file version.

```
>>> m.version
'F03_0024'
```

2.2 MtkGrid

class MtkGrid

Grid from file.

```
>>> g = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
```

attr_get(attr_name)

Get a grid attribute.

```
>>> g.attr_get('Block_size.resolution_x')
1100
```

attr_list

List of attribute names.

```
>>> g.attr_list
['Block_size.resolution_x', 'Block_size.resolution_y', 'Block_size.size_x', 'Block_size.size_y']
```

field_dims(field_name)

Returns a list of tuples of the extra dimension names and sizes. If field_name doesn't have extra dimensions an empty list is returned.

```
>>> g.field_dims('Blue Radiance')
[]
```

field(field_name)

Return MtkField.

```
>>> g.field('Blue Radiance')
<MisrToolkit.MtkField object at 0x15137a0>
```

field_list

List of field names.

```
>>> g.field_list
['Blue Radiance/RDQI', 'Blue Radiance', 'Blue RDQI', 'Blue DN', 'Blue Equivalent Reflectance']
```

native_field_list

List of native field names (excludes derived fields).

```
>>> g.native_field_list
['Blue Radiance/RDQI']
```

grid_name

Grid name.

```
>>> g.grid_name
'BlueBand'
```

resolution

Resolution of grid in meters.

```
>>> g.resolution
1100
```

2.3 MtkField

class MtkField

fieldname Field from grid.

```
>>> f = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
```

data_type

Data type of field.

```
>>> f.data_type
'uint16'
```

field_name

Field name.

```
>>> f.field_name
'Blue Radiance'
```

fill_value

Fill value.

```
>>> f.fill_value
65515
```

read(region)

Read data from field by specifying a MtkRegion. Returns a MtkDataPlane.

```
>>> r = MtkRegion(37, 50, 55)
>>> f.read(r)
<MisrToolkit.MtkDataPlane object at 0x18a0400>
```

Note: The MtkField read method always return a 2-D data plane buffer. Some fields in the MISR data products are multi-dimensional. In order to read one of these fields, the slice to read needs to be specified. A bracket notation on the fieldname is used for this purpose. For example RetrAppMask[0][5].

Additional dimensions can be determined using MtkGrid field_dims method or by referencing MISR Data Product Specification (DPS) Document. The actually definition of the indices are not described in the MISR product files and thus not described by the MISR Toolkit. These will have to be looked up in the MISR DPS. All indices are 0-based.

read (*start_block, end_block*)

Reads native fields and returns a 3-D NumPy array for the block range. The blocks are not assembled and are just stacked on top of each other. The CoordQuery functions can be used to map or convert geographic coordinates into block, line and sample which correspond to the data returned by this function.

This function is provided as an alternative to reading blocks one at a time using a MtkRegion.

```
>>> f.read(50, 55)
array([[65515, 65515, 65515, ..., 65515, 65515, 65515],
       [65515, 65515, 65515, ..., 65515, 65515, 65515],
       [65515, 65515, 65515, ..., 65515, 65515, 65515],
       ...,
       [65515, 65515, 65515, ..., 65515, 65515, 65515],
       [65515, 65515, 65515, ..., 65515, 65515, 65515],
       [65515, 65515, 65515, ..., 65515, 65515, 65515]]], dtype=uint16)
```

Note: The block index returned by the CoordQuery functions are 1-based and are referenced to the entire MISR path. The 3-D array returned by this method is referenced to your 1-based start block and Python uses 0-based indexing, so adjust the block index accordingly.

2.4 MtkRegion

class MtkRegion()

Construct a new MtkRegion object.

```
>>> r = MtkRegion()
```

class MtkRegion (*path, start_block, end_block*)

Construct a new MtkRegion object by path and block range.

```
>>> r = MtkRegion(37, 50, 60)
```

class MtkRegion (*ulc_lat, ulc_lon, lrc_lat, lrc_lon*)

Construct a new MtkRegion object by upper left corner, lower right corner.

```
>>> r = MtkRegion(40.0, -120.0, 30.0, -110.0)
```

class MtkRegion (*ctr_lat, ctr_lon, lat_extent, lon_extent, extent_units*)

Construct a new MtkRegion object by latitude, longitude in decimal degrees, and extent in specified units.

The extent_units argument is a case insensitive string that can be set to one of the following values:

1. "degrees", "deg", "dd" for degrees;

2. "meters", "m" for meters;
3. "kilometers", "km" for kilometers; and
4. "275m", "275 meters", "1.1km", "1.1 kilometers" for pixels of a specified resolution per pixel.

```
>>> r = MtkRegion(35.0, -115.0, 1.5, 2.0, "deg")
>>> r = MtkRegion(35.0, -115.0, 5000.0, 8000.0, "m")
>>> r = MtkRegion(35.0, -115.0, 2.2, 1.1, "km")
>>> r = MtkRegion(35.0, -115.0, 45.0, 100.0, "275m")
>>> r = MtkRegion(35.0, -115.0, 35.0, 25.0, "1.1km")
```

class MtkRegion (*path, ulc_som_x, ulc_som_y, lrc_som_x, lrc_som_y*)

Construct a new MtkRegion object by Path and SOM X/Y of upper left corner and lower right corner in meters.

```
>>> r = MtkRegion(27, 15600000.0, -300.0, 16800000.0, 2000.0)
```

block_range (*path*)

Return block range that covers the region for the given path.

```
>>> r.block_range(37)
(59, 67)
```

center

Center coordinate of the region in degrees.

```
>>> r.center
(35.0, -115.0)
```

extent

Extent of the region in meters.

```
>>> r.extent
(1113195.4314, 1113195.4314)
```

path_list

List of paths that cover the region.

```
>>> r.path_list
[33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45]
```

snap_to_grid (*path, resolution*)

Snap a region to a MISR grid based on path number and resolution.

```
>>> r.snap_to_grid(37, 1100)
<MISRToolkit.MtkMapInfo object at 0x1897a00>
```

2.5 MtkDataPlane

class MtkDataPlane

Contains data and map information.

```
>>> r = MtkRegion(37, 50, 50)
>>> d = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
```

data()

Returns a NumPy of the data in the plane.

```
>>> d.data()
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)
```

mapinfo()

Returns a MtkMapInfo for the data in the plane.

```
>>> d.mapinfo()
<MisrToolkit.MtkMapInfo object at 0x189b200>
```

2.6 MtkMapInfo

class MtkMapInfo

Contains map information, and supports map queries.

```
>>> map_info = MtkRegion(37, 35, 36).snap_to_grid(37, 1100)
```

create_latlon()

Create a latitude array and a longitude array in decimal degrees.


```
>>> map_info.create_latlon()
(array([[ 69.84396725,  69.84125375,  69.83853597, ...,  67.95214383,
         67.94751056,  67.94287408],
       [ 69.83449667,  69.83178431,  69.82906768, ...,  67.94344755,
         67.93881609,  67.9341814 ],
       [ 69.82502575,  69.82231453,  69.81959905, ...,  67.93475031,
         67.93012064,  67.92548775],
       ...,
       [ 67.43810121,  67.43564618,  67.43318732, ...,  65.72400133,
         65.71978632,  65.71556829],
       [ 67.42855667,  67.42610255,  67.4236446 , ...,  65.71509301,
         65.71087952,  65.70666301],
       [ 67.41901189,  67.41655868,  67.41410164, ...,  65.70618396,
         65.701972 ,  65.697757 ]]),

array([[ -99.94798275,  -99.92052402,  -99.89307216, ...,  -87.01432073,
        -86.9911815 ,  -86.96805169],
       [ -99.95583948,  -99.92839213,  -99.90095165, ...,  -87.0266458 ,
        -87.00351269,  -86.98038898],
       [ -99.96368936,  -99.93625339,  -99.90882428, ...,  -87.03896147,
        -87.01583448,  -86.99271687],
       ...,
       [-101.7397708 , -101.71492204, -101.69007822, ...,  -89.85775933,
        -89.83612005,  -89.81448799],
       [-101.74615355, -101.72131411, -101.69647961, ...,  -89.86799566,
        -89.84636208,  -89.82473572],
       [-101.75253136, -101.72770125, -101.70287606, ...,  -89.87822477,
        -89.85659689,  -89.83497623]]))
```

end_block

End block number.

```
>>> map_info.end_block
36
```

geo

MtkGeoRegion object.

```
>>> map_info.geo
<MisrToolkit.MtkGeoRegion object at 0xe458>
>>> map_info.geo.ctr
(67.848508,-94.442838)
```

latlon_to_ls (*lat, lon*)

Lat and Lon to Line Sample.

```
>>> map_info.latlon_to_ls(68.36,-97.74)
(120.52, 120.55)
```

ls_to_latlon (*line, sample*)

Line and Sample To Lat and Lon.

```
>>> map_info.ls_to_latlon(120,120)
(68.36, -97.74)
```

ls_to_somxy (*line, sample*)

Convert Line and Sample values to Som X and Som Y coordinates. line and sample can be either scalar values or numarrays.

```
>>> map_info.ls_to_somxy(120,120)
(12380500.0, 677600.0)
```

nline

Number of lines.

```
>>> map_info.nline
256
```

nsample

Number of samples.

```
>>> map_info.nsample
512
```

path

Path number.

```
>>> map_info.path
37
```

pixelcenter

Pixel Center.

```
>>> map_info.pixelcenter
True
```

pp

MtkProjParam object.

```
>>> map_info.pp
<MisrToolkit.MtkProjParam object at 0x18ade00>
```

resfactor

Resfactor.

```
>>> map_info.resfactor
4
```

resolution

Resolution.

```
>>> map_info.resolution
1100
```

som

MtkSomRegion object.

```
>>> map_info.som
<MisrToolkit.MtkSomRegion object at 0xe338>
>>> map_info.som.ctr
(12388750.0, 826650.0)
```

somxy_to_ls (*somx*, *somy*)

Som X and Som Y To Line and Sample.

```
>>> map_info.somxy_to_ls(12380500.0, 677600.0)
(120.0, 120.0)
```

start_block

Start block number.

```
>>> map_info.start_block
35
```

2.7 MtkTimeMetaData

class MtkTimeMetaData

Contains time metadata information, and supports calculating pixel time.

```
>>> time_metadata = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_0029058_AA_F
```

Note: Time metadata is stored in the L1B2 Ellipsoid product, it is not available in any other product. The pixel time varies according to camera by approximately 7 minutes. To get the average or center pixel acquisition time, it is recommended to use the time metadata from the AN camera.

camera

Camera for which time metadata applies.

```
>>> time_metadata.camera
'AA'
```

coeff_line

Line transform coefficients.

```
>>> time_metadata.coeff_line
array([[ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00]],

      [[ 2.06016807e+03,  2.31347217e+03],
       [ 9.89420163e-01,  9.89470448e-01],
       [ 2.32523204e-02,  2.23010430e-02],
       [ 1.03023724e-05,  1.03026080e-05],
       [-3.72946711e-06, -3.72871997e-06],
       [-6.93100473e-13, -1.58848849e-11]],

      [[ 2.56679833e+03,  2.82013585e+03],
       [ 9.89537006e-01,  9.89609347e-01],
       [ 2.13693245e-02,  2.04218034e-02],
       [ 1.03134648e-05,  1.03101703e-05],
       [-3.77586865e-06, -3.69789446e-06],
       [-5.35620372e-11, -1.11540278e-11]],

      ...,

      [[ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00]],

      [[ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00]],

      [[ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00]]])
```

end_block

End block number.

```
>>> time_metadata.end_block
140
```

number_line

Number of lines.

```
>>> time_metadata.number_line
array([[ 0, 0],
       [256, 256],
       [256, 256],
       [256, 256],
       ...,
       [ 0, 0],
       [ 0, 0],
       [ 0, 0],
       [ 0, 0]])
```

number_transform

Number of transforms.

```
>>> time_metadata.number_transform
array([0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

path

Path number.

```
>>> time_metadata.path
37
```

pixel_time (*som_x*, *som_y*)

Calculate pixel time at Som X, Som Y.

```
>>> time_metadata.pixel_time(10153687.5, 738787.5)
'2005-06-04T18:06:07.656501Z'
```

ref_time

Reference time.

```
>>> time_metadata.ref_time
[['', ''], ['2005-06-04T17:58:13.127920Z', '2005-06-04T17:58:13.127920Z'], ... ['', ''], ['2005-06-04T17:58:13.127920Z', '2005-06-04T17:58:13.127920Z'], ...]]
```

som_ctr_x

SOM X center coordinates.

```
>>> time_metadata.som_ctr_x
array([[ 0., 0.],
       [ 128., 384.],
       [ 640., 896.],
       ...,
       [ 0., 0.],
       [ 0., 0.],
       [ 0., 0.]])
```

som_ctr_y

SOM Y center coordinates.

```
>>> time_metadata.som_ctr_y
array([[ 0.,  0.],
       [1024., 1024.],
       [1024., 1024.],
       ...,
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

start_block

Start block number.

```
>>> time_metadata.start_block
1
```

start_line

Starting line.

```
>>> time_metadata.start_line
array([[ 0,  0],
       [ 0, 256],
       [512, 768],
       ...,
       [ 0,  0],
       [ 0,  0],
       [ 0,  0]])
```

2.8 MtkReProject

class MtkReProject

Contains ReProjection Functionality.

```
>>> reproj = MtkReProject()
```

create_geogrid()

Create a geo-grid array in decimal degress for lat/lon.

```
>>> reproj.create_geogrid(40,-120,30,-110,0.25,0.25)
(array([[ 40. ,  40. ,  40. , ...,  40. ,  40. ,  40. ],
       [ 39.75,  39.75,  39.75, ...,  39.75,  39.75,  39.75],
       [ 39.5 ,  39.5 ,  39.5 , ...,  39.5 ,  39.5 ,  39.5 ],
       ...,
       [ 30.5 ,  30.5 ,  30.5 , ...,  30.5 ,  30.5 ,  30.5 ],
       [ 30.25,  30.25,  30.25, ...,  30.25,  30.25,  30.25],
       [ 30. ,  30. ,  30. , ...,  30. ,  30. ,  30. ]]),

array([[-120. , -119.75, -119.5 , ..., -110.5 , -110.25, -110. ],
       [-120. , -119.75, -119.5 , ..., -110.5 , -110.25, -110. ],
       [-120. , -119.75, -119.5 , ..., -110.5 , -110.25, -110. ],
       ...,
       [-120. , -119.75, -119.5 , ..., -110.5 , -110.25, -110. ],
       [-120. , -119.75, -119.5 , ..., -110.5 , -110.25, -110. ],
       [-120. , -119.75, -119.5 , ..., -110.5 , -110.25, -110. ]]))
```

resample_cubic_convolution()

Resample source data at the given coordinates using interpolation by cubic convolution.

```
>>> srcdata = numpy.ones((128,512), dtype=numpy.float32) * 0.04
>>> datashape = srcdata.shape
>>> srcmask = numpy.ones(datashape, dtype=numpy.uint8)
>>> a = -0.5;
>>> regrshape = tuple((float(dimen) * abs(a)) for dimen in datashape)
>>> lines = numpy.tile(numpy.linspace(4.1,((10*regrshape[0]) + 4.1), regrshape[1]), (regrshape[0],))
>>> samples = numpy.tile(numpy.linspace(4.1,((10*regrshape[1]) + 4.1), regrshape[0]), (regrshape[1],))
>>> myproj.resample_cubic_convolution(srcdata, srcmask, lines, samples, a )
(array([[ 0. ,  0. ,  0. , ...,  0. ,  0. ,  0. ],
       [ 0.04,  0.04,  0.04, ...,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04, ...,  0.04,  0.04,  0.04],
       ...,
       [ 0.04,  0.04,  0.04, ...,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04, ...,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04, ...,  0.04,  0.04,  0.04]], dtype=float32),

array([[0, 0, 0, ..., 0, 0, 0],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       ...,
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1]], dtype=int8))
```

resample_nearest_neighbor()

Performs nearest neighbor resampling.

```

>>> srcdata = numpy.ones((128,512), dtype=numpy.float32) * 20
>>> datashape = srcdata.shape
>>> regrshape = tuple( (float(dimen) * 0.5) for dimen in datashape)
>>> lines = numpy.tile(numpy.linspace(4.1,((10*regrshape[0]) + 4.1), regrshape[1]), (regrsh
>>> samples = numpy.tile(numpy.linspace(4.1,((10*regrshape[1]) + 4.1), regrshape[0]), (regr
>>> myproj.resample_nearest_neighbor(srcdata, lines, samples)
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)

```

transform_coordinates()

Transforms latitude/longitude coordinates into line/sample coordinates.

```

>>> r = Mtk.MtkRegion(39, 51, 52)
>>> mapinfo = r.snap_to_grid(39, 275)
>>> (lat, lon) = reproj.create_geogrid(49.0, -113.0, 47.5, -114.0, 0.02, 0.02)
>>> reproj.transform_coordinates(mapinfo, lat, lon)
(array([[475.7129 , 474.96417, 474.21405, ..., 512.7551 , 512.07776,
        511.39905],
       [483.72412, 482.97504, 482.22458, ..., 520.7843 , 520.1066 ,
        519.4276 ],
       [491.73526, 490.98584, 490.23502, ..., 528.8135 , 528.13544,
        527.45605],
       ...,
       [ -1.      , -1.      , -1.      , ..., -1.      , -1.      ,
        -1.      ],
       [ -1.      , -1.      , -1.      , ..., -1.      , -1.      ,
        -1.      ],
       [ -1.      , -1.      , -1.      , ..., -1.      , -1.      ,
        -1.      ]], dtype=float32), array([[313.62213 , 318.89313 , 324.1639 , ..., 39.2
        49.779778],
       [314.75842 , 320.03152 , 325.30438 , ..., 40.2419 , 45.526855,
        50.811584],
       [315.89478 , 321.16998 , 326.44495 , ..., 41.269634, 46.556675,
        51.843494],
       ...,
       [ -1.      , -1.      , -1.      , ..., -1.      , -1.      ,
        -1.      ],
       [ -1.      , -1.      , -1.      , ..., -1.      , -1.      ,
        -1.      ],
       [ -1.      , -1.      , -1.      , ..., -1.      , -1.      ,
        -1.      ]], dtype=float32))

```

2.9 MtkRegression

class MtkRegression

Contains Regression Functionality.

```

>>> regr = MtkRegression()

```


downsample()

Downsamples data by averaging pixels.

```
>>> r = MtkRegion(37, 50, 60)
>>> m = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
>>> g = m.grid('BlueBand')
>>> f = m.grid('BlueBand').field('Blue Radiance')
>>> srcdata = f.read(r).data()
>>> srcmask = numpy.ones((srcdata.shape[0],srcdata.shape[1]), dtype= numpy.uint8)
>>> sizefactor = 2
>>> rsmpdata, rsmpmask = regr.downsample(srcdata,srcmask,sizefactor)
>>> print rsmpdata[0][90]
315.341
```

linear_regression_calc()

Uses linear regression to fit data.

```
>>> x = numpy.linspace(2,12,num = 5)
>>> y = numpy.linspace(4,24,num = 5)
>>> ysigma = numpy.linspace(0.1,0.6,num = 5)
>>> regr.linear_regression_calc(x,y,ysigma)
(array([ 0.00000000e+000, -2.32035723e+077,  2.15126649e-314,
        1.06717989e-287,  3.38503263e+125]),
 array([ 2.00000000e+000, -2.68679011e+154,  2.15168741e-314,
        2.15170772e-314,  2.32035723e+077]),
 array([ 1.00000000e+000,  3.11109040e+231,  2.15175217e-314,
        6.94154000e-310, -3.11109040e+231]))
```

smooth_data()

Smooths the given array with a boxcar average.

```
>>> r = MtkRegion(37, 50, 60)
>>> m = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
>>> g = m.grid('BlueBand')
>>> f = m.grid('BlueBand').field('Blue Radiance')
>>> srcdata = f.read(r).data()
>>> srcmask = numpy.ones((srcdata.shape[0],srcdata.shape[1]), dtype= numpy.uint8)
>>> line_width = 3
>>> sample_width = 3
>>> regr.smooth_data(srcdata,srcmask,line_width,sample_width)
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)
```

upsample_mask()

Upsamples a mask by nearest neighbor sampling.

```

>>> srcmask = numpy.ones((1408,624), dtype= numpy.uint8)
>>> size_factor = 2
>>> regr.upsample_mask(srcmask,size_factor)
array([[1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)

```

coeff_calc()

Calculates linear regression coefficients for translating values.

```

>>> r = MtkRegion(37, 50, 60)
>>> m = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
>>> g1 = m.grid('BlueBand')
>>> f1 = g1.field('Blue Radiance')
>>> data1 = f1.read(r).data()
>>> mask1 = numpy.ones((data1.shape[0],data1.shape[1]), dtype= numpy.uint8)
>>> g2 = m.grid('GreenBand')
>>> f2 = g2.field('Green Radiance')
>>> data2 = f2.read(r).data()
>>> sigma2 = numpy.tile((numpy.linspace(0.1,0.6,data2.shape[0])), (data2.shape[1],1)).transpose(1,0)
>>> sigma2 = sigma2.astype(numpy.float32)
>>> mask2 = numpy.ones((data2.shape[0],data2.shape[1]), dtype= numpy.uint8)
>>> mapinfo = f2.read(r).mapinfo()
>>> sizefactor = 2
>>> regr.coeff_calc(data1, mask1, data2, sigma2, mask2, sizefactor, mapinfo)
(<MisrToolkit.MtkRegCoeff object at 0x102e99930>, <MisrToolkit.MtkMapInfo object at 0x7fde1...

```

apply_regression()

Applies regression to given data.

```

>>> r = MtkRegion(37, 50, 60)
>>> m = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
>>> g1 = m.grid('BlueBand')
>>> f1 = g1.field('Blue Radiance')
>>> data1 = f1.read(r).data()
>>> mask1 = numpy.ones((data1.shape[0],data1.shape[1]), dtype= numpy.uint8)
>>> g2 = m.grid('GreenBand')
>>> f2 = g2.field('Green Radiance')
>>> data2 = f2.read(r).data()
>>> sigma2 = numpy.tile((numpy.linspace(0.1,0.6,data2.shape[0])), (data2.shape[1],1)).transpose(1,0)
>>> sigma2 = sigma2.astype(numpy.float32)
>>> mask2 = numpy.ones((data2.shape[0],data2.shape[1]), dtype= numpy.uint8)
>>> mapinfo = f2.read(r).mapinfo()
>>> sizefactor = 2
>>> regr_coeff, regr_mapinfo = regr.coeff_calc(data1, mask1, data2, sigma2, mask2, sizefactor)
>>> regr.apply_regression(data1, mask1, mapinfo, regr_coeff, regr_mapinfo)
(array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        ...,
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32),
array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=uint8))

```

resample_reg_coeff()

Resamples regression coefficients at each pixel.

```

>>> r = MtkRegion(37, 50, 60)
>>> m = MtkFile('../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_O029058_AA_F03_0024.hdf')
>>> g1 = m.grid('BlueBand')
>>> f1 = g1.field('Blue Radiance')
>>> data1 = f1.read(r).data()
>>> mask1 = numpy.ones((data1.shape[0],data1.shape[1]), dtype= numpy.uint8)
>>> g2 = m.grid('GreenBand')
>>> f2 = g2.field('Green Radiance')
>>> data2 = f2.read(r).data()
>>> sigma2 = numpy.tile((numpy.linspace(0.1,0.6,data2.shape[0])), (data2.shape[1],1)).transpose(1,0)
>>> sigma2 = sigma2.astype(numpy.float32)
>>> mask2 = numpy.ones((data2.shape[0],data2.shape[1]), dtype= numpy.uint8)
>>> mapinfo = f2.read(r).mapinfo()
>>> sizefactor = 2
>>> regr_coeff, regr_mapinfo = regr.coeff_calc(data1, mask1, data2, sigma2, mask2, sizefactor)
>>> target_map_info = MtkRegion(37, 50, 60).snap_to_grid(37, 1100)
>>> regr.resample_reg_coeff(regr_coeff, regr_mapinfo, target_map_info)
<MisrToolkit.MtkRegCoeff object at 0x10a623a30>

```

3 Containers

All the classes in this sections are returned from other functions.

3.1 MtkBlockCorners

class MtkBlockCorners

Block Corners.

block

1-based indexed tuple of MtkGeoBlock.

end_block

End block number.

path

Path number.

start_block

Start block number.

3.2 MtkGeoBlock

class MtkGeoBlock

Geographic Block Coordinates.

block

Block number.

ctr

MtkGeoCoord containing center coordinate.

llc

MtkGeoCoord containing lower left coordinate.

lrc

MtkGeoCoord containing lower right coordinate.

ulc

MtkGeoCoord containing upper left coordinate.

urc

MtkGeoCoord containing upper right coordinate.

3.3 MtkGeoCoord

class MtkGeoCoord

Geographic Coordinates.

lat

Latitude in decimal degrees.

lon

Longitude in decimal degrees.

3.4 MtkGeoRegion

class MtkGeoRegion
Geometric Region.

ctr
MtkGeoCoord containing center coordinate.

llc
MtkGeoCoord containing lower left coordinate.

lrc
MtkGeoCoord containing lower right coordinate.

ulc
MtkGeoCoord containing upper left coordinate.

urc
MtkGeoCoord containing upper right coordinate.

3.5 MtkSomCoord

class MtkSomCoord
SOM Coordinates.

x
SOM X coordinate.

y
SOM Y coordinate.

3.6 MtkSomRegion

class MtkSomRegion
SOM Region.

ctr
MtkSomCoord containing center coordinate.

lrc
MtkSomCoord containing lower right coordinate.

path
Path number.

ulc
MtkSomCoord containing upper left coordinate.

3.7 MtkProjParam

class MtkProjParam
This object contains projection parameters.

lrc
Lower right corner.

nblock
Number of blocks.

nline
Number of lines.

nsample
Number of samples.

path
Path number.

projcode
Projcode.

projparam
Projection parameters.

resolution
Resolution.

reloffset
Reloffset.

spherecode
Sphere code.

ulc
Upper left corner.

zonecode
Zone code.

3.8 MtkRegCoeff

class MtkRegCoeff
Contains regression coefficient information.

```
>>> regr_coeff = MtkRegCoeff()
```

```
>>> regr_coeff, regr_mapinfo = regr.coeff_calc(data1, mask1, data2, sigma2, mask2, sizefact)
```

valid_mask()
Returns a NumPy of valid mask for regression coefficients.

```
>>> regr_coeff.valid_mask()
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

slope()
Returns a NumPy of slopes for regression coefficients.

```
>>> regr_coeff.slope()
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)
```

intercept()

Returns a NumPy of intercepts for regression coefficients.

```
>>> regr_coeff.intercept()
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)
```

correlation()

Returns a NumPy of correlation for regression coefficients.

```
>>> regr_coeff.correlation()
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32)
```

4 Functions

4.1 CoordQuery

bls_to_latlon (*path, resolution_meters, block, line, sample*)

Convert from Block, Line, Sample, to Latitude and Longitude in Decimal Degrees.

```
>>> bls_to_latlon(189, 275, 47, 12.5, 50.5)
(55.161373, 16.435317)
```

bls_to_somxy (*path, resolution_meters, block, line, sample*)

Convert from Block, Line, Sample, to SOM Coordinates.

```
>>> bls_to_somxy(230, 1100, 69, 100.2, 89.9)
(17145919.996, 222089.993)
```

latlon_to_bls (*path, resolution_meters, lat, lon*)

Convert decimal degrees latitude and longitude to block, line, sample.

```
>>> latlon_to_bls(160, 1100, 57.1, 65.7)
(45, 19.5214, 207.8861)
```

latlon_to_somxy (*path, lat, lon*)

Convert decimal degrees latitude and longitude to SOM X, SOM Y.

```
>>> latlon_to_somxy(160, 57.1, 65.7)
(13677973.731, 686274.716)
```

path_block_range_to_block_corners (*path, start_block, end_block*)

Compute block corner coordinates in decimal degrees of latitude and longitude for a given path and block range.

```
>>> path_block_range_to_block_corners(37, 50, 53)
<MisrToolkit.MtkBlockCorners object at 0x189f800>
```

path_to_projparam (*path, resolution*)

Get projection parameters.

```
>>> path_to_projparam(160, 275)
<MisrToolkit.MtkProjParam object at 0x1897a00>
```

somxy_to_bls (*path, resolution_meters, somx, somy*)

Convert SOM X, SOM Y to block, line, sample.

```
>>> somxy_to_bls(230, 1100, 17145920.0, 222090.0)
(69, 100.2, 89.899)
```

somxy_to_latlon (*path, somx, somy*)

Convert SOM X, SOM Y to decimal degrees latitude and longitude.

```
>>> somxy_to_latlon(230, 17145920.0, 222090.0)
(26.7376, -54.1496)
```

4.2 FileQuery

find_file_list (*searchdir, product, camera, path, orbit, version*)

Find files in directory tree, using regular expressions.

```
>>> find_file_list('../Mtk_testdata/in/', 'GRP_ELLIPSOID_GM', '.*', '037', '029058', 'F03_0024')
['MISR_AM1_GRP_ELLIPSOID_GM_P037_0029058_AA_F03_0024.hdf', '../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_0029058_AA_F03_0024.hdf']
```

make_filename (*basedir, product, camera, path, orbit, version*)

Given a base directory, product, camera, path, orbit, version make file name.

```
>>> make_filename('../Mtk_testdata/in/', 'GRP_ELLIPSOID_GM', 'AA', 37, 29058, 'F03_0024')
'../Mtk_testdata/in/MISR_AM1_GRP_ELLIPSOID_GM_P037_0029058_AA_F03_0024.hdf'
```


4.3 OrbitPath

latlon_to_path_list (*lat, lon*)

Get list of paths that cover a particular latitude and longitude.

```
>>> latlon_to_path_list(66.121646, 89.263022)
[7, 8, 9, 10, 11, 12, 13, 14, 146, 147, 148, 149, 150, 151, 152, 153, 154]
```

orbit_to_path (*orbit*)

Given orbit number return path number.

```
>>> orbit_to_path(29058)
37
```

orbit_to_time_range (*orbit*)

Given orbit number return time range. Time format: YYYY-MM-DDThh:mm:ssZ (ISO 8601)

```
>>> orbit_to_time_range(32467)
('2006-01-24T19:56:53Z', '2006-01-24T21:35:46Z')
```

path_time_range_to_orbit_list (*path, start, end*)

Given path and time range return list of orbits on path. Time format: YYYY-MM-DDThh:mm:ssZ (ISO 8601)

```
>>> path_time_range_to_orbit_list(37, '2002-02-02T02:00:00Z', '2002-05-02T02:00:00Z')
[11350, 11583, 11816, 12049, 12282, 12515]
```

time_range_to_orbit_list (*start, end*)

Given start time and end time return list of orbits. Time format: YYYY-MM-DDThh:mm:ssZ (ISO 8601)

```
>>> time_range_to_orbit_list('2005-02-02T02:00:00Z', '2005-02-02T03:00:00Z')
[27271, 27272]
```

time_to_orbit_path (*time*)

Given time return orbit number and path number. Time format: YYYY-MM-DDThh:mm:ssZ (ISO 8601)

```
>>> time_to_orbit_path('2005-02-02T02:00:00Z')
(27271, 104)
```

4.4 UnitConv

dd_to_deg_min_sec (*dd*)

Convert decimal degrees to unpacked degrees, minutes, seconds.

```
>>> dd_to_deg_min_sec(65.55)
(65, 32, 60.0)
```

dd_to_dms (*dd*)

Convert decimal degrees to packed degrees, minutes, seconds.

```
>>> dd_to_dms(65.55)
65032060.0
```

dd_to_rad(*dd*)

Convert decimal degrees to radians.

```
>>> dd_to_rad(65.55)
1.144063
```

deg_min_sec_to_dd(*deg, min, sec*)

Convert unpacked degrees, minutes, seconds to decimal degrees.

```
>>> deg_min_sec_to_dd(65, 33, 0.001)
65.55
```

deg_min_sec_to_dms(*deg, min, sec*)

Convert unpacked Degrees, minutes, seconds to packed.

```
>>> deg_min_sec_to_dms(65, 33, 0.001)
65033000.001
```

deg_min_sec_to_rad(*deg, min, sec*)

Convert unpacked degrees, minutes, seconds to radians.

```
>>> deg_min_sec_to_rad(65, 33, 0.001)
1.144063
```

dms_to_dd(*dms*)

Convert packed degrees, minutes, seconds to decimal degrees.

```
>>> dms_to_dd(65033000.010)
65.55
```

dms_to_deg_min_sec(*dms*)

Convert packed degrees, minutes, seconds to unpacked.

```
>>> dms_to_deg_min_sec(65033012.0)
(65, 33, 12.0)
```

dms_to_rad(*dms*)

Convert packed degrees, minutes, seconds to Radians.

```
>>> dms_to_rad(65033000.010)
1.144063
```

rad_to_dd(*rad*)

Convert radians to decimal degrees.

```
>>> rad_to_dd(1.1440634)
66.55
```

rad_to_deg_min_sec(*rad*)

Convert radians to unpacked degrees, minutes, seconds.

```
>>> rad_to_deg_min_sec(1.14406333)
(65, 33, 0.00109)
```

rad_to_dms (*rad*)

Convert radians to packed degrees, minutes, seconds.

```
>>> rad_to_dms(1.14406332)
65032059.999
```

4.5 Util

cal_to_julian (*year, month, day, hour, min, sec*)

Convert calendar date to Julian date.

```
>>> cal_to_julian(2005, 12, 23, 18, 33, 18)
2453728.2731249998
```

datetime_to_julian (*datetime*)

Convert date and time in ISO 8601 format to Julian date.

```
>>> datetime_to_julian('2005-12-23T18:33:18Z')
2453728.2731249998
```

julian_to_cal (*julian_date*)

Convert Julian date to calendar date.

```
>>> julian_to_cal(2453728.2731249998)
(2005, 12, 23, 18, 33, 18)
```

julian_to_datetime (*julian_date*)

Convert Julian date to date and time in ISO 8601 format.

```
>>> julian_to_datetime(2453728.2731249998)
'2005-12-23T18:33:18Z'
```

parse_fieldname (*field_name*)

Parses extra dimensions from fieldnames.

version ()

MISR Toolkit version.

```
>>> version()
'1.5.1'
```